

**A USER INTERFACE FOR FEATURE-PAIR BASED
DESIGN AND ANALYSIS OF MECHANICAL
ASSEMBLIES**

By

PENGFEI CAI

Bachelor of Engineering

Tsinghua University

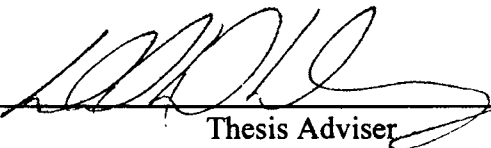
Beijing, P. R. China

1990

**Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1994**

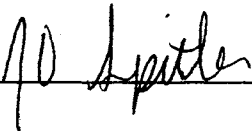
A USER INTERFACE FOR FEATURE-PAIR BASED
DESIGN AND ANALYSIS OF MECHANICAL
ASSEMBLIES

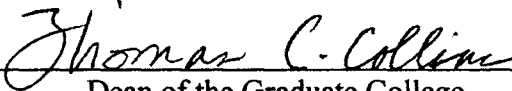
Thesis Approved:



Thesis Adviser







Dean of the Graduate College

ACKNOWLEDGMENTS

I would like to take this opportunity to express my sincere appreciation to my adviser Dr. R. D. Delahoussaye for his encouragement and advice throughout my graduate program. Many thanks also go to Dr. Spitler and Dr. Misawa for serving on my graduate committee.

My parents, Qigui Cai and Yuehua Jiao encouraged me all the way and helped me keep the end goal constantly in sight. Thanks go to my grandfather Zhiping Lu for supporting me during the study. My wife Tong Zhao provided moral support and was a real believer in my abilities. I extend a sincere thank you to all of these people.

TABLE OF CONTENTS

Chapter		Page
I.	INTRODUCTION.	1
	Research Problem	1
	Objective.	2
II.	LITERATURE REVIEW	6
	History of Geometric Modeller	6
	Feature Based Design Overview.	11
	Review of Feature Modeling Work.	11
	Definition of Feature.	12
	Feature Based Design System Functional	
	Requirements	14
	Assembly Modeller Using Virtual Link	15
	Assembly Modeler Using Assembly Features	20
	Dynamic Models for Simulation	21
III.	USER INTERFACE DESIGN FOR A FEATURE PAIR	
	BASED DESIGN SYSTEM	26
	System Overview.	26
	Data Structure Used in The System.	27
	Concept Of Feature Pair.	32
	User Interface Design.	43
	Event Driven Menu System	43
	Selection Algorithm.	44
	Handles for Editing.	46
	Modify a Feature Pair	49
	Visual Effect of the Graphics	53
V.	DYNAMIC MODEL AND DISTRIBUTED	
	COMPUTATION FOR SIMULATION	54
	Use Feature Pair To Speed Up The Simulation.	54
	Distributed Computation For Simulation.	59

Chapter		Page
VI	CONCLUSION AND RECOMMENDATION	65
	Conclusion	65
	Recommendation	66
	REFERENCES	67

LIST OF TABLES

Table		Page
I.	Number of Calculation in General Method.	56
II.	Number of Calculation with Rigid Body Concept	57
III.	Number of Calculation with Feature Pair Concept.	58

LIST OF FIGURES

Figure		Page
1.	Schematic for the Feature Based Design System	3
2.	Schematic of the Distributed Computation	5
3.	Evolution of Geometric Modellers	7
4.	Comparison of Different Kind of Modellers	8
5.	Tree Structure for Assembly With Virtual Link	17
6.	Constraint Change of an Object	22
7.	Calculate the Reaction Force	22
8.	An Assembly with Two Simple Features	28
9.	The Object Linked List for Figure 8.	30
10.	The Tree Data Structure for Figure 8.	31
11.	An Assembly with Two Components	34
12.	The Design Procedure for Figure 11	35
13.	The Data Structure Representation of Figure 11	36
14.	The Design Procedure Using Feature Pair Concept	37
15.	The Data Structure of Figure 11 with Feature Pair Definition	38
16.	A Complicated Mechanical System with Several Components	40
17.	The Schematic for Figure 16	41
18.	The Assembly Representation of the System	42

Figure		Page
19.	The Selection Algorithm Demonstration	45
20.	The Handles of a Circle or Pin	47
21.	Handles of a Block	47
22.	Handles of a Polygon or a Curve	48
23.	Handles of a Rigid Body	49
24.	Relative Movement in a Feature Pair	51
25.	Move a Feature Pair with No Relative Movement	52
26.	Move All Connected Feature As a Rigid Body	52
27.	Move a Feature Pair in a Rigid Body	53
28.	A Simple Mechanical System	55
29.	Speed Up the Simulation with Feature Pair	55
30.	Server-Client Model	61
31.	Flow Chart of the Server	62
32.	Flow Chart of the Client	63

CHAPTER I

INTRODUCTION

Research Problem

In modern industry, there is a general need for better quality designs of manufactured products and their parts. It is widely believed that 70% or more of the life cycle cost of a product is determined during the early design process (DeFazio, 1988). It is very important to develop design tools and aids that will make designs more effective.

Commercial CAD/CAM software today supporting engineering functions, such as mass properties calculations, interference checking, geometry definition for drafting, and finite elements are currently implemented at a level that require large amounts of human interaction. Also, current CAD systems can only capture an instance of the design. The system then has no general basis for managing constraints later, when changes are made. The result is a system which requires the designer to attend to all details associated with the design changes. Even if features can be successfully extracted from the geometry later (Kumer, 1988), the designer's intent will have been lost. So further automation of engineering tasks is not possible unless there are fundamental changes made in the underlying geometric modeling schemes.

Recently, the concept of using features for design and manufacturing has gained much attention and research interest. That is because a large amount of information about features automatically comes from the feature database. Very little input is needed from the designer. Feature-based design captures the designer's intent about a single part. A CAD system with the designer's intention included in the representations of the in-progress

design will be a helpful tool, not just a graphics recording systems. This allows users to use minimal information input to get a high quality design output.

In this thesis, based on the concept of features, we will develop a concept of feature pair. Features capture the designer's intent about a rigid body. Feature pairs capture the designer's intent about the assembly. We will develop a graphical user interface (GUI) to support the feature pair based design. Because the feature pairs represent a mechanical system at assembly level, feature-based design becomes a special case for the feature pair based design. That means the GUI supports the feature-based design too. Our GUI allows the user to use feature and feature pairs to construct a mechanical system. Then with a simple dynamic model, we will do a simulation on the system. In the simulation, we will implement distributed computation to speed up the calculation. The schematic of the system is shown in figure 1.

Objective

Although many research works have been reported, there is no commercial product based entirely on feature-based design. The purpose of this thesis is to implement the concept of feature pair based design to a design system, which is written in C language running on Silicon Graphics. The reason we selected the Silicon Graphics is that Motif and GL (a graphics library) mixed model is supported on operating system 2.1. Motif provides a very good user interface and GL provides the quality and high speed graphics. For future work, we will incorporate a solid modeller to the system. GL can provide a good lighting model and Non-Uniform Rational B-Spline (NURBS) function which is very difficult to implement in other tool boxes. But the calculation speed of Silicon Graphics is not fast. So when there are many features in the system, the simulation becomes slow. To solve this problem, we use the distributed computation method to speed it up. We use the RS6000

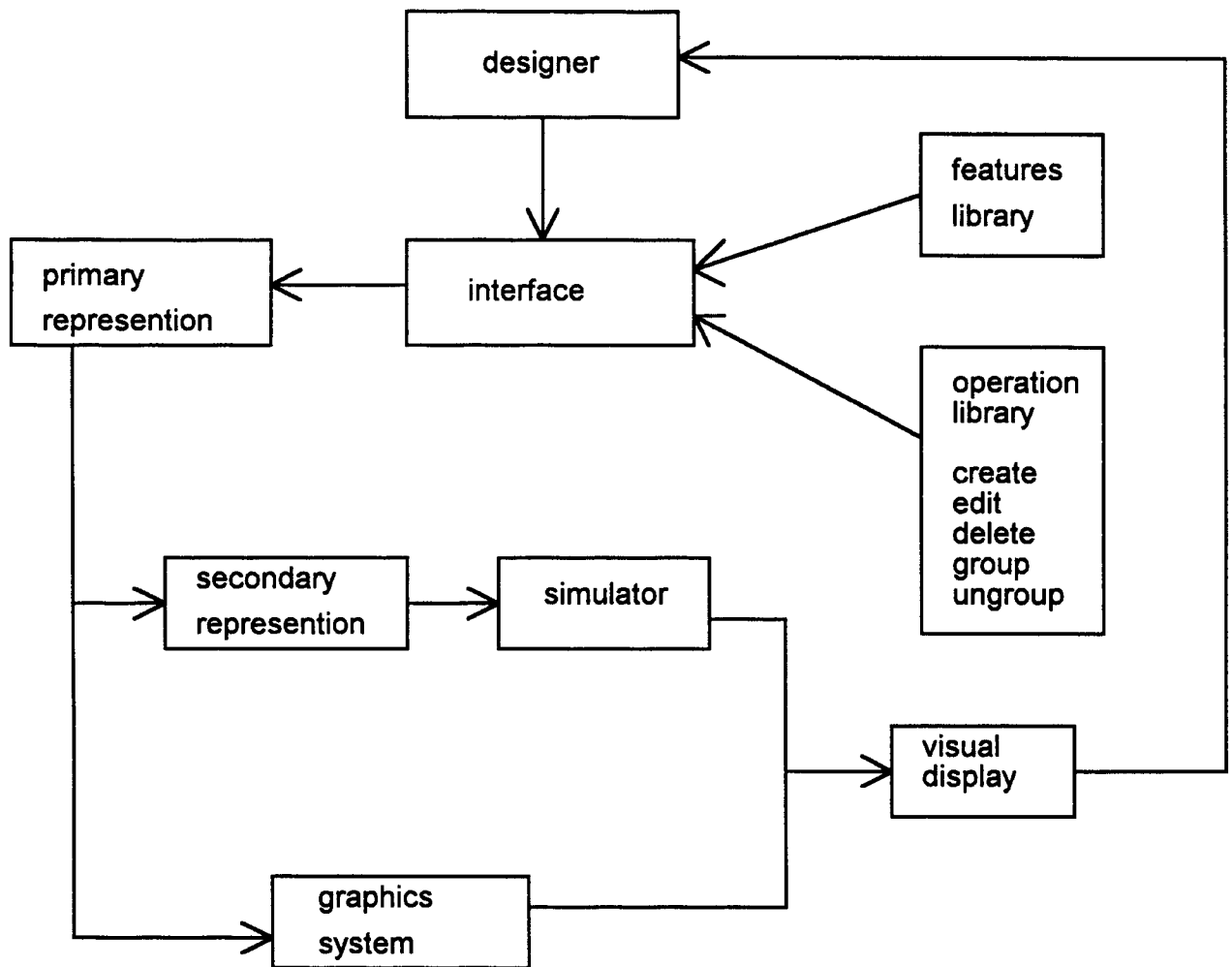


Figure 1. Schematic for the Feature Based Design System

which has a faster floating point processor to do the computation work. Then we use the interprocess communication to transfer the result back to Silicon Graphics. For interprocess communication, we use BSD socket as the transfer protocol. The schematic of the distributed computation is shown in figure 2.

The modeling system is developed in 2D space. This simplifies the developing work. It can handle many mechanical systems, especially the planer system. But it is ambiguous in representing some problems. So it is better to represent the features in 3D space.

From the above discussion, we know we will develop a system including the following parts:

- 1) A graphical user interface which allows users to create, modify, and delete the features and feature pairs interactively or group features into a rigid body rapidly.
- 2) A hierarchical data structure to support the feature based design and feature pair based design.
- 3) Use the feature pair concept to a dynamic simulation system.
- 4) A distributed computation protocol using interprocess communication with BSD socket, and implement it to speed up the dynamic simulation.

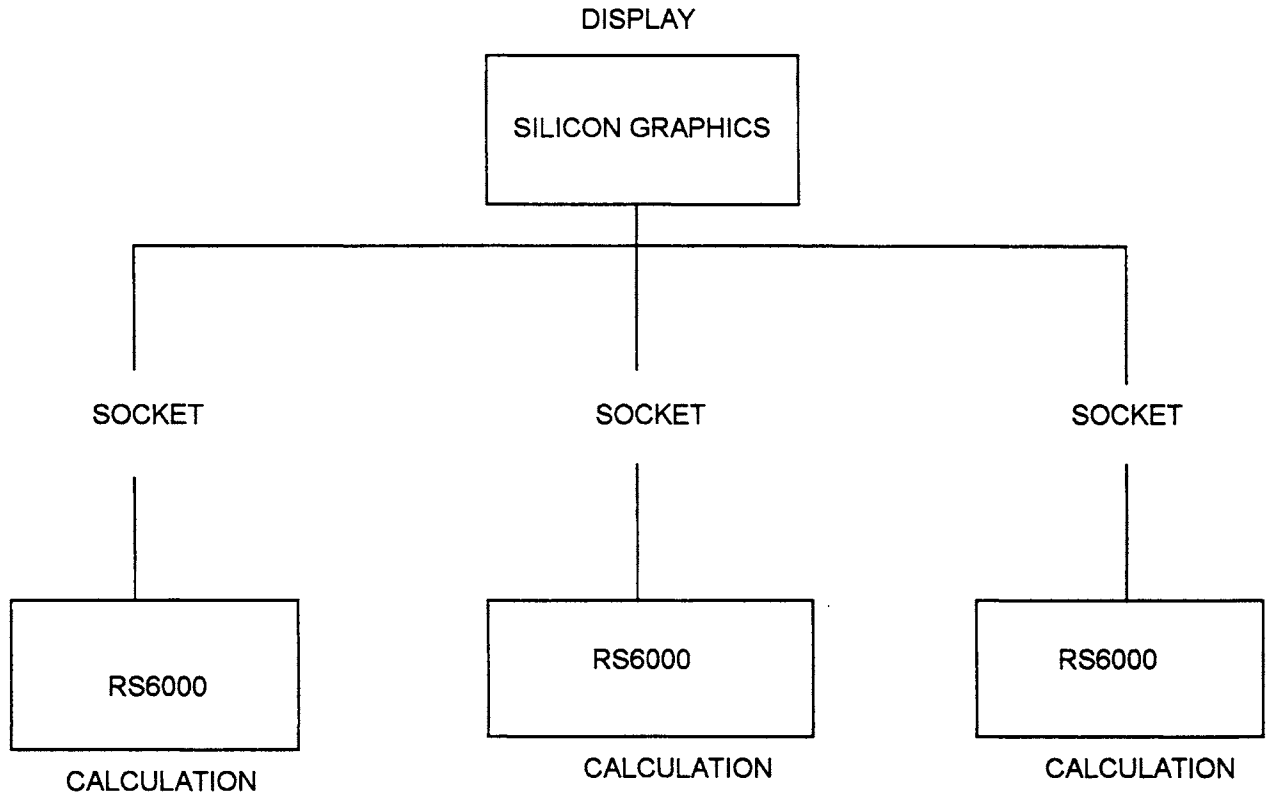


Figure 2. Schematic of the Distributed Computation

CHAPTER II

LITERATURE REVIEW

History Of Geometric Modeller

Geometric modellers have undergone tremendous changes over the twenty years of their existence, as depicted in figure 3 (Shah, 1988a). Each phase of this evolution has been propelled by the inadequacies of contemporary techniques available at that time. The fundamental difference between each generation of modellers is the information level supported. Early wireframe systems supported only two types of geometric entities (points and straight line segments), which made them unsuitable for anything but drafting of simple parts. These systems have evolved into today's Boundary Representation (B-Rep) modellers, which support all geometric and topological entities right up to the volume level. Figure 4 (Shah, 1988a) compares the capabilities of each modeller type to the application needs that were responsible for further development.

In the early 1980's, a solid model was considered to be an "informationally complete" representation of the physical object of which some properties, like volume or surface area, could be calculated automatically without human help. It is unambiguous compared with the previous modellers, but it has some problems.

Solid modellers can only be used to define the nominal geometry. Information regarding surface finish, tolerances, material properties, surface conditions, etc., are

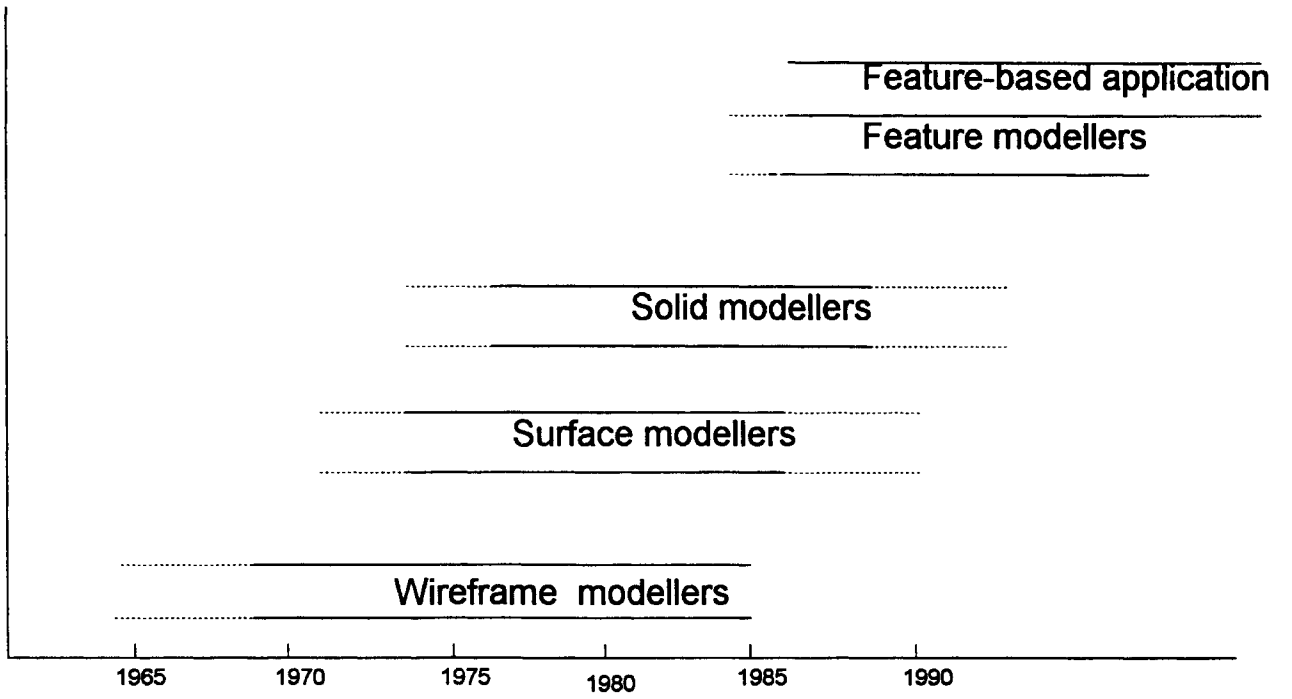


Figure 3. Evolution of Geometric Modellers

Modeller	Application	Limitations
2D wireframe	Drafting	No viewing transfer Not a solid representation Nonsense geometry
3D wireframe	Drafting	Not a real solid representation Nonsense geometry Ambiguous images
Surface modeller	Graphic art Visualization of complex geometry NC programming	Nonsense geometry Not a real solid representation
Solid modellers	Constructive building with Boolean operator Mass properties calculate	Nominal geometry Low level of abstraction cannot be used to drive automated applications

Figure 4. Comparison of Deferent Kind of Modellers

important parts of the definition of mechanical parts, but these cannot be incorporated in a solid modeller database.

Solid modellers store data in terms of low level entities such as vertices, edges, faces, etc., or binary trees that contain primitives and Boolean operators. It is difficult to extract the engineering meaning of this data from the solid model database. For example, it is not a trivial task to recognize from the database that a body is rotational and has a stepped-through hole. Yet, this is the level of information that many application programs operate on. The feature recognition community is currently engaged in extracting this information from solid models. The method, however, is laborious and so far has only been demonstrated for simple features.

Shah (1988a) thought "solid modellers do not provide an environment conducive to creativity in design." The user must 'know' the design before a solid modeller can be used because modifications are difficult. Suppose a part has a hole on which a groove has been defined. The user wishes to reduce the hole diameter without changing the shape or depth of the groove. If the user was using a solid modeller, he would need to first 'fill' the space of the hole and groove, then create primitives for the cylinder and torus with new dimensions, and then carry out the necessary Boolean operations to create the new object. According to Shah "This illustrates that solid modellers do not allow changes to be made easily and all parameters have to be input explicitly when creating each primitive object, even if the parameter has been defined previously." Design is an iterative process and as such requires numerous alterations in the model before a satisfactory solution is obtained.

Feature-based design provides a solution to these problems. For the database problem, other than to provide a single form feature database which describes the geometry and topology of an object, feature-based design provides precision, material, and several feature databases which can be used to define objects. For the low level representation problem, feature-based design provides a design with features mechanism. Instead of extracting the feature after the design phase, the feature is used to design

objects. All the information about the features is stored in the database. If a body of a part is rotational and has a stepped hole in it, we just define it as "feature rotational body + a stepped hole." It is easy and clear. The only requirement is that the definitions of "rotational body" and "stepped hole" are in your database. Feature-based design also provides a very good design environment. The user only needs to take care of the feature he wants to use, and the system knows how to generate the edge table and surface table according to the feature parameter user input. Referring back to the part which has a hole on which a groove has been defined. When the user wishes to reduce the hole diameter without changing the shape or depth of the groove, he needs to input the new diameter to the system. The system knows how to deal with the geometry and topology itself (Shah, 1988b).

From the above discussion, we know that feature-based design provides a complete database, a high level representation of a single rigid body, and a very good design environment. It is also a new approach to the old problem of attempting to link CAD with CAM and process planning. The features approach constrains the designer or process planner to working with a set of features which have significance for either design, analysis, or manufacturing. Instead of using a model consisting of graphics primitives, the designer is asked to use a set of features from which manufacturing operations can be derived. When both the designer and process planner have finished the design and process plan, more information has been entered into the model than if the part had been created on a traditional CAD system. However, the designer and planner have entered the information not at the geometric level but at a higher level (Unger and Roy, 1988).

Feature Based Design Overview

Review of Feature Modelling Work

In the past decade, a large amount of research work has been done in the area of feature-based design. Several systems have been developed for research purposes. Dixon (1988) has worked on several feature based design systems in the area of extrusions, injection moldings, and castings. He tried to apply artificial intelligence to mechanical engineering. He viewed features as a way to represent 3D design and manufacturing geometry. He also feels that features are a good approach for representing design knowledge in AI systems.

Shah (1988b) developed an expert form feature modeling shell. In this shell three modellers worked separately; form feature modellers, precision features modellers, and material features modellers. After the form features are created, other features are created independently, and then networked together. He also implemented group technology for formalizing geometric reasoning for GT coding and building a GT knowledge base using the generic mapping shell.

Turner and Anderson (1988) have developed an object-oriented approach to feature based design. They implement the approach to Quick Turnaround Cell project. The project is to create an environment where one-of-a-kind parts can be designed and manufactured "immediately" with human input required at only the design and machine setup stages. The QTC tightly couples feature based design, automatic fixturing, process planning, NC code generation, and visual inspection into a manufacturing system that quickly produces parts using little operator knowledge or intervention.

Faux (1988) reconciled earlier work done within CAM-I (Computer Aided Manufacturing International) in the area of form features and dimensional tolerances. A feature hierarchy was defined along with sets of rules. The lowest level indivisible units are termed primitive features that are sets of connected faces that obey certain declared rules.

These, in turn, could be used for defining compound features and patterns. This decomposition into low-level elements is suited to tolerancing.

Hirschtick (1986) has developed a system called "The Extrusion Advisor." It is a system designed to advise a user in the domain of aluminum extrusions. The system examines a geometric model of a part to be manufactured by an extrusion process. It extracts features and then identifies features which will cause manufacturing difficulties. Hirschtick uses "characteristic patterns" to find an instance of a feature. The geometric definition of the part is searched for one or more characteristic patterns. Several of these patterns are combined to form a characteristic pattern set or feature. He concludes that feature based design and feature extraction will both be necessary. He calls feature based design "feature descriptor modeling ."

Definition Of Features

The term "feature" has been used for more than a decade. But the definitions of it differ because each researcher considers different ideas significant. Some researchers are interested in designing with features, such as Dixon (1988). He defined a feature as "a geometric form or entity that is used in reasoning in one or more design or manufacturing activities (i.e., fit, function, manufacturability evaluation, analysis interfacing, tool and die design, inspectability, serviceability, etc.)." Here he considered the features as a geometric form because he worked on a design-with-feature system at that time. Henderson (1988) defined a feature as "a part 'characteristic,' including material type, functionality and other pieces of descriptive information." Henderson devised a feature recognizer program to produce a list of primitive features from boundary representation model. Hirschtick (1986) said "a feature of a geometric model in a given context is a descriptor of that model whose presence and/or size is relevant within the given context." Hummel (1986), who worked in the area of feature representation for process planning, told us "features are regions of the

part that have some degree of manufacturing significance. Put another way, features form reoccurring geometric and technologic patterns for which the process engineer has acquired years of manufacturing experience."

From the above, we can see that the definition of a feature is dependent upon its use. This is the unfortunate aspect of current feature definitions. A features representation is not unique. A designer and a manufacturer working with a feature model must agree on the same feature definitions and interpretations. If one considers current CAD models, a large amount of the information on the drawings is not geometric and requires consistent interpretations between manufacturing and design. Interpretation is required because some manufacturing attributes cannot be passed geometrically. So we need a unique definition on features and on both design and manufacturing that can easily work with the model. This is not the purpose of this thesis, but we can find some common points in these definitions. All definitions mentioned that features are a "geometric entity." They all assume that features are combined to form parts or other objects. And they all imply that features provide a higher level model of the object than does a traditional CAD geometric model.

In this thesis, we will use Shah's (1988a) definition. It is convenient to group product information into sets based on the meaning of the information. These information sets will be called features. He also defines it informally as "recurring patterns of information related to a part's description." Shah divided the features into several types, and defined them separately.

Form Features: These are groups of geometric entities that define attributes of a part's nominal size and shape. The grouping of geometric entities to define form features depends on the task for which they are to be used, e.g. design, manufacturing, etc.

Precision Features: These are acceptable deviations from the nominal geometry. Included in this set are dimensional tolerances and surface finish. Designers specify a range

of acceptable limits, datum planes, and material condition in order to ensure that manufactured parts will assemble properly and function as desired.

Material Features: These specify material types, grades, properties, heat treatment, surface treatments, etc. There are, of course, many other logical sets, such as assembly features, functional features, analysis features, etc.

Through this thesis, we will work on form features. So when we mention features in the following chapters, we mean form features except those accompanied by a special description.

Feature Based Design System Functional Requirements

Shah (1988a) pointed out a feature based design system must satisfy the following requirements:

All fundamental information sets related to mechanical products must be supported. These were identified previously as form features, precision features, and material features.

The particular features needed depend on the product being designed; thus the needs of every organization are going to be different. It is necessary, therefore, to provide a mechanism whereby each organization can define its own set of features.

The product definition system (the feature modeller) must provide an attractive environment for creating, manipulation, modifying, and deleting product models. It should be possible to define and instance form, precision, and material features individually and define relationships between form features as well as intrafeature relationships. For example, defining tolerances after creating the nominal geometry involves creating a network between form features, precision features, and geometric entities.

If it is recognized that the interpretation of features is application specific, it is necessary to create a feature mapping system to transform the design database into an

application specific database. The mechanism to do this must also be immensely flexible so that it can be set up to support any application that uses the product database.

Finally, the system must have a feature database. In general, the feature geometry database may contain the following types of information:

- (1) A list of all instances of all form features and their locations and dimensions.
- (2) A generic definition of each feature in terms of generic parameters, geometry, and topology.
- (3) A list of relationships between form features (dependencies and adjacencies).
- (4) A geometric model of the entire part.
- (5) Material properties and other nongeometric attributes.
- (6) Tolerances and surface finish where appropriate.

Assembly Modeller Using Virtual Link

Several systems are available for automatic assembling for research propose. But most of them need a preprocessor to create a geometric model. The preprocessor could be a feature based design system or just a common solid modeller or B-Rep modeller.

Ko and Lee (1987) developed a system which can automatically generate an assembly hierarchical structure from a B-Rep model created by a separate B-Rep modeller. In their work, an assembly is described by its components and their relationship in an assembly, especially the mating conditions. From the mating conditions of the components, each component is located at a specific vertex of a hierarchical tree. In the system, they try to minimize the user's input. But the users still need to input the mating condition between mating components. And the system can support only four mating types: against, fit, tight fits, and contact.

The virtual link data structure used in Ko and Lee's system was developed by Lee and Gossard (1985). As shown in figure 5, an assembly may be composed of many

subassemblies and components in a hierarchy, and can be stored in a graph structure using the concept of a 'virtual link' created by them. In the assembly data structure, any mating pair of two subassembly, two components, or one subassembly and one component is connected by a virtual link. If more than two components are mutually related, several virtual links can be used so that every pair of mating components occupies one virtual link. They define the virtual link as "the complete set of information required to describe the relationship (e.g. rigid attachment, rotational constraints) and the mating features between the mating pair." They use "mating features" here, but it is not the feature in the feature based design. From the structure definition we can see that mating feature means face or line. In figure 5, an assembly located at the top node is composed of one or more pairs of subassemblies, where every pair is connected by a virtual link. A subassembly may be composed of several pairs of subassemblies and components which are connected into pairs by virtual links, and so on. In this way, the terminal nodes of the assembly graph will be the components of the assembly, and the component data for each component should be connected to these terminal nodes.

This data structure represents the mating geometry at a low level. But it has its own advantage. First, the assembly can be automatically updated for modifications in the size of any component in the assembly, in contrast to other structure which requires explicit modification of transformation matrices according to the size change of some components. Second, the structure stores only possible assemblies. If an assembly of components which cannot be assembled together is stored in the data structure, there is not a set of transformation matrices of all the components which satisfies all the mating conditions. This can be detected when the transformation matrices are computed from mating conditions. Third, the assembly data structure has a hierarchy very similar to that of assembly procedures for components. Therefore, it is expected that the simulation for assembly procedures can be achieved with this structure. Finally, dimension chains in an

assembly can be derived from mating feature information so that tolerance analysis on the assembly can be performed.

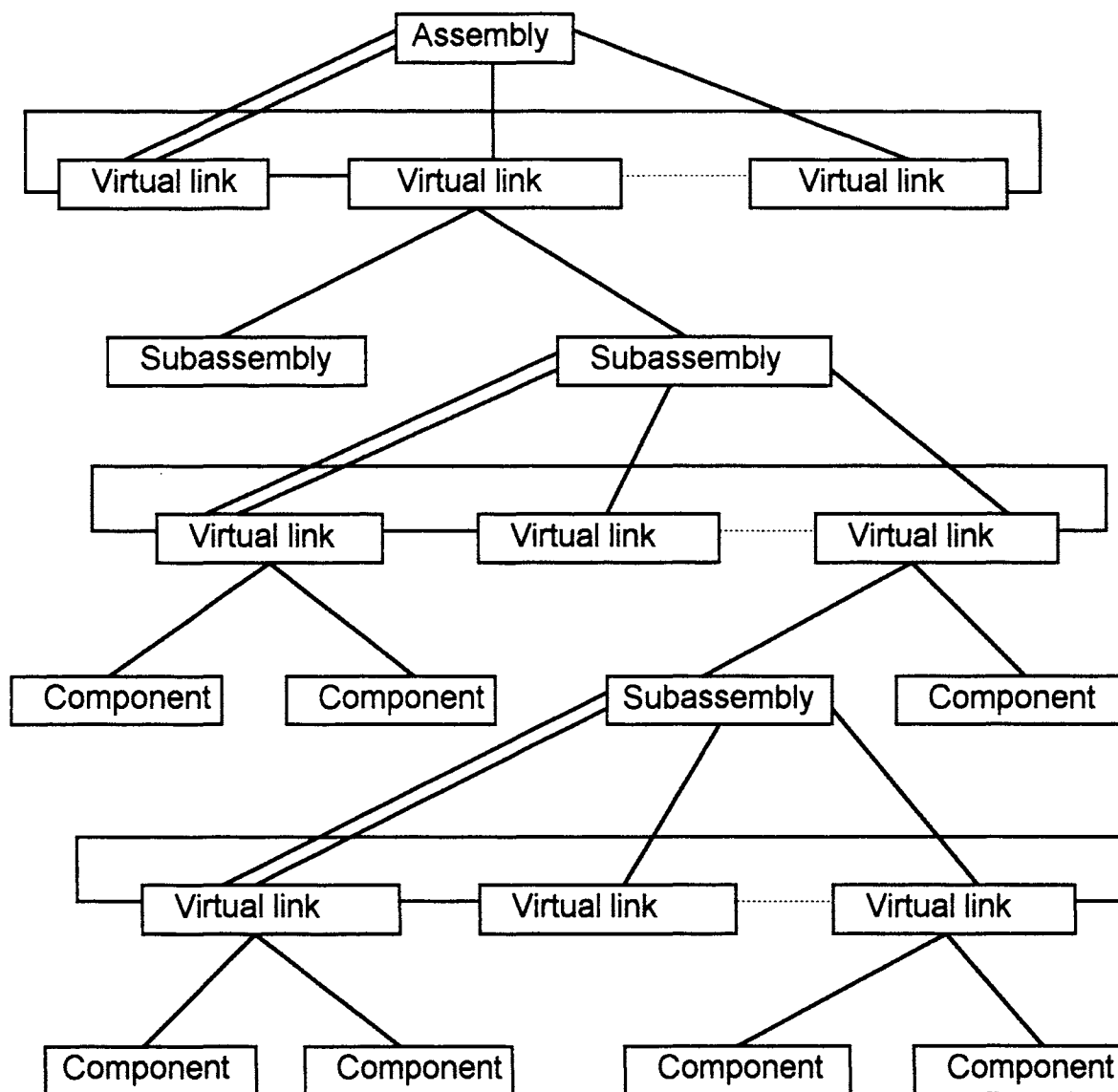


Figure 5. Tree Structure for Assembly with Virtual Link

The structures they define are as follows:

ASSEMBLY:

Name of the assembly

Pointer to the first virtual link of the immediate lower level

VIRTUAL-LINK

Pointer to the next virtual link

Pointer to an assembly or a subassembly of the immediate upper level

Pointer to the starting location in MATING-FEATURE

Number of records in MATING-FEATURE related to the virtual link

Pointer to a component or a subassembly of the pair

Pointer to the other component or the other subassembly of the pair

Relationship provided by this virtual link, e.g. rigid attachment, conditional attachment, translational constraint, and rotational constraint

SUBASSEMBLY:

Name of subassembly

Pointer to subassembly or assemble of the immediate upper level

Pointer to the first virtual link of the immediate lower level

Stores the derived transformation from the assembly or the subassembly of the immediate upper level

Mass properties of the subassembly

MATING-FEATURE:

Type of mating, e.g. 'against' and 'fits'

Mating feature (planar face or centerline)

Mating feature (planar face or centerline)

Clearance between mating features

Here we focus on the assembly, so we did not show the component structure in their work.

Their system is a very typical one of today's assembly modellers. They can successfully generate assembly hierarchy in most situation, but they have some problems.

The geometric modeller is separated with the assembly modeller. In the design phase, the designer cannot give any information about the assembly to the system so the system does not have any idea about the assembly. If the geometric model fails in the assembly phase, the designer has to go back to the geometric modeller to make changes. This is not an effective methodology. It would be convenient for the user if the assembly could be represented at design phase and if the system could finish the interference checking automatically. If some errors exist, the designer could make changes immediately.

The user needs to input the mating condition between components, and the system doesn't provide an effective way for describing it. The system supports only four types of mating conditions. They are not enough to describe a general mechanical system. Sometimes it is difficult to describe a mating situation by only one kind of mating condition. Different users may give different mating conditions to the same problem because it is hard to tell how tight belongs to "tight fits." This kind of fussy logic gives the system and user some ambiguity. Although the user input is tried to be minimized, the input of mating condition will still bother the user.

The data structure used in their system had some geometrical information in it for the interference checking. But the information is in the geometric level, such as face or centerline. In practice, the mating condition actually is described by the tolerance. And tolerance should be defined at the feature level. If two features in a component share the same centerline, the system will have a problem describing the mating situation clearly.

Assembly Modeller Using Assembly Features

Shah (1992) used the concept of "assembly feature" in an assembly modeller. He defined the assembly feature as "an association between two form features which are on different parts." Assembly features encode mutual constraints on mating features' shape, dimension, position, and orientation.

His assembly modeller has two shells. The first is called Set-Up mode. In this mode an assembly feature library is built on the top of the form feature library of individual features. It allows the user to define assembly features with generic form features and provides constraint rules on them. The second mode is called "Modeling mode." This mode is "for the design of assemblies using feature-based descriptions of piece parts modeled in the feature modeller." It allows the user to define assembly relationship between two parts (rigid bodies) with the defined assembly features.

Shah's "assembly feature" is almost the same as our "feature pair" concept. His also use mating features to establish the assembly hierarchy. The difference is we concentrate on design with feature pair, capture the designer's intent in the design phase. His system tries to recognize mating pairs after the design phase. He associates the constraints between the assembly feature with the library in rule format. This is the biggest success in his system. It allows the user to change it in the design phase instead of hard code the constraint in the program. Although the rule defining and rule firing takes a little overhead, it does give the user some flexibility. If the system provides some default constraint, the disadvantage would be reduced. For the further development, it is easier to be developed toward an artificial intelligence system which can handle generic assemblies if a proper knowledge base is established. Then the work in the Set-Up mode could be done by the system automatically. The drawback of the system is that it separates form feature modeller and assembly modeller into two shells. It is not convenient for the user to come back and forth in the two shells.

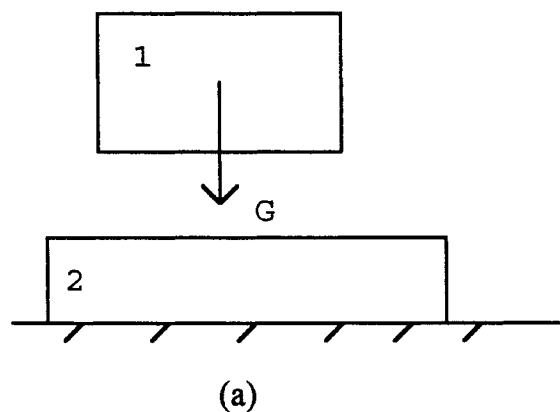
Dynamic Model For Simulation

Dynamic Model Using Penetration Between Two Rigid Body to Calculate Reaction Force

Due to the changes in the kinematics constraints, many mechanical systems are described by discontinuous equations of motion. In our system, we use a simple dynamic model which can detect constraint changes automatically. This model was developed by Hong in his master thesis (1991).

When objects move in the system, the constraints will change. In figure 6 (a) object 1 is a free object. The only force acting on it is the gravity G . As figure 6 (b) shows, object 1 falls down on object 2. There is a reaction force R that acts on it. So the total force in object 1 is equal to the gravity G plus the reaction force R .

Now we try to use the collision of the objects to detect the constraint change between the two objects. If two polygons collide, the reaction force should be created. We first calculate the maximum collision distance. Figure 7 shows the collision. AB is the maximum collision distance of the two objects. CD is the width of the collision.



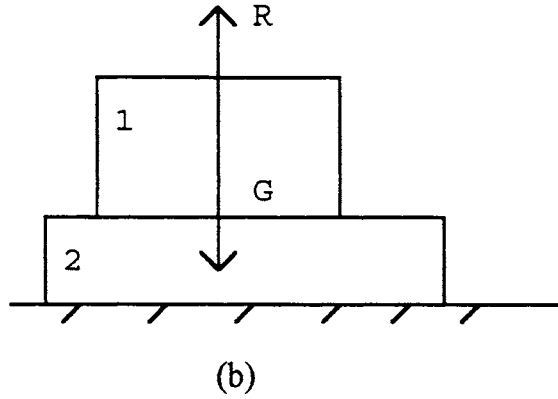


Figure 6. Constraints Change of an Object

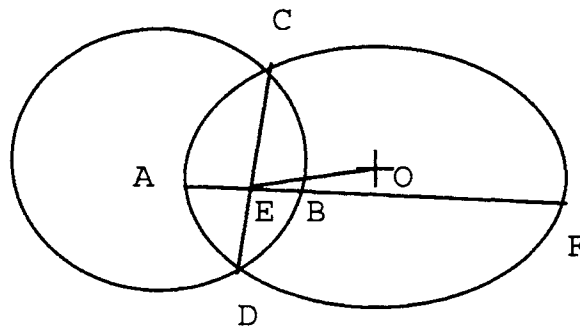


Figure 7. Calculate the Reaction Force

In figure 7, CD is the intersection line between the two objects. E is the center of CD. AF is the normal of CD, it intersects with object O at A and F. O is the center of object O. We take the shorter one in AE and EF as the force direction. In this example it is AE. If the angle between EO and EA is larger than 90, the force direction is AE for object O. Otherwise, the force direction is EA. The force for the other object is the same amount and in the reverse direction.

We have

$$F_1 = k_1 * x_1 * w$$

$$F_2 = k_2 * x_2 * w$$

Where k_1 and k_2 are stiffness of object 1 and object 2. x_1 and x_2 are deformation of object 1 and object 2, w is the width of the collision. Then we get:

$$x_1 = \frac{F_1}{k_1 * w}$$

$$x_2 = \frac{F_2}{k_2 * w}$$

So:

$$x = x_1 + x_2 = \frac{F_1}{k_1 * w} + \frac{F_2}{k_2 * w}$$

Since

$$F_1 = F_2$$

So:

$$F_1 = F_2 = \frac{k_1 * k_2 * w * x}{k_1 + k_2}$$

Our system is designed for 2D problem at this time, so Newton's second law is used to generate the motion equation.

$$\Sigma F = m * a$$

$$\Sigma M = J * \epsilon$$

So we can get the acceleration and the angular acceleration from the force F1 and F2. Using the continuous time formulas below:

$$v = v_0 + a*t$$

$$s = s_0 + v_0*t + \frac{1}{2}*a*t^2$$

$$\omega = \omega_0 + \varepsilon*t$$

$$\alpha = \alpha_0 + \omega_0*t + \frac{1}{2}*\varepsilon*t^2$$

We can get the discrete time formulas as follows:

$$v_{t+h} = v_t + a_t*h$$

$$s_{t+h} = s_t + v_t*h + \frac{1}{2}*a_t*h^2$$

$$\omega_{t+h} = \omega_t + \varepsilon_t*h$$

$$\alpha_{t+h} = \alpha_t + \omega_t*h + \frac{1}{2}*\varepsilon_t*h^2$$

We can see that if we know the previous step's a_t , v_t , s_t , ε_t , ω_t , and α_t , we can calculate the status of the next step.

Dynamic Model Which Automatically Generate and Numerically Solve the Differential Equations of Motion

Differential equations of motion are the traditional and natural way to describe a dynamic system. This model cannot handle a general mechanical system due to the unpredictable constraint changes. The constraint changes must be anticipated by the user.

So how to automatically predict and detect the kinematics constraint changes is the main barrier of using this model.

Gilmor and Cipra (1991) presented a strategy to predict the constraint changes automatically. As they pointed out, if a point on a rigid body is constrained to remain in contact with a surface of another rigid body, the point is restricted to move in the direction of the direction tangent of the surface. For a line segment on a rigid body, if both endpoints of it contacts a line segment on another rigid body, then the two bodies are line contact. In a planar system, each body is represented by line segments. So they simplify a planar system to several point to line constraints. A constraint between bodies may be formed as the result of a collision where the collision is viewed as a boundary interference but not a boundary penetration (like Hong's method). If two bodies collide and remain in contact, the one or more point to line constraints are added or deleted to the equations of motion. The constraints change in the following conditions:

1. Constraint formation due to boundary interference.
2. Constraint break due to the restitution of an impact.
3. Constraint break due to the insufficient closing force.
4. Constraints break due to relative sliding.

This model is designed to handle a planar system. It works better than Hong's model, especially for line constraints problems.

CHAPTER III

USER INTERFACE DESIGN OF A FEATURE PAIR BASED DESIGN SYSTEM

System Overview

Last chapter we discussed the concept and history of feature-based design in detail. In this chapter we will propose a concept of feature pair on the top of feature-based design. It gives the user more alternatives in the design phase, and allows the system to represent assembly at a higher level.

The following will be implemented in the system:

- (1) Develop a data structure which can support a hierarchical group of objects, and support feature pair design.
- (2) Develop a concept of feature pair and implement it in a dynamic simulation system.
- (3) Develop an event driven graphical user interface in which the user can create, manipulate, modify, and delete features and feature pairs very easily.
- (4) Implement a simple dynamic model in the simulation. This model should be able to detect the constraint change automatically.
- (5) Implement the distributed computation to the system to speed up the simulation.

Data Structure Used In The System

The feature pair is developed on the top of features. It gives the user more opportunities to represent the assembly at the design phase. Furthermore, it allows the user to modify any feature, even after the assembly is constructed. To have a better understanding about the feature pair concept, we need to introduce the hierarchy data structure used in the system first.

Feature Data Structure

In the system, each feature has its own representation. The data structures used in the system are as follows.

```
typedef struct p3d{
    double *point;
    struct p3d *next;
} P3D;

typedef struct Slot{
    double *center;
    double *dir;
    struct p3d polygon;
} MY_SLOT;

typedef struct Pin{
    double *center;
    double r;
    struct p3d polygon;
} MY_PIN;

typedef MY_PIN MY_HOLE;
typedef MY_SLOT MY_BLOCK;
```

In the pin, slot, block, and hole data structures, there is a field "polygon". That is the secondary representation of the feature. The primary representation give a full description of the geometry, but it is an abstraction of a mathematical model. For example, the pin is stored as the center and radius. This saves space. But it is not convenient to calculate the intersection between features, because we have to construct a different intersection model for each two different features. Here when we need to calculate intersection, we first convert features to polygon, and store them in the polygon field. This conversion is done by the system automatically. After calculation, we release the space, so we need only one model to calculate the intersection; that is the intersection between two polygons.

Group Data Structure

In the real world, most parts consist of more than one feature. So we need to group features into a rigid body. Figure 8 shows an assembly with two very simple parts. One consists of a block feature and two hole features. The other is a pin feature. The data structure for representing a rigid body is as follows. We call the rigid body a group.

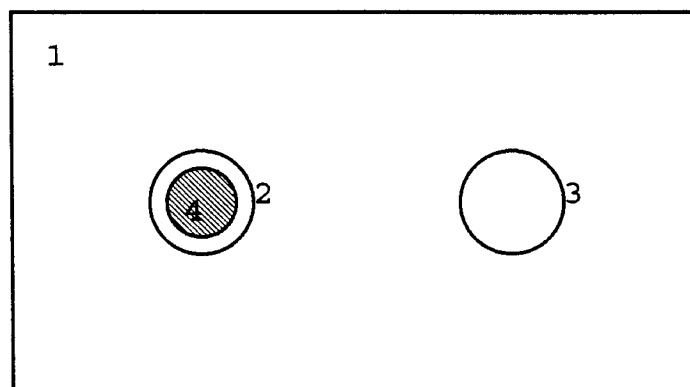


Figure 8 An Assembly with Two Simple Features

```
typedef struct Group{
    double *left, *right;
    MY_OBJECT *head;
```

```

}    MY_GROUP;

```

The right and left field is for group handle. We will discuss that later. The "head" field is the head of an object linked list. This object linked list is the list of objects in the rigid body. The objects could also be a group of features. So this is a hierarchy data structure.

Object Data Structure

Object is a feature or a feature group.

```

typedef Object{
    int id;
    int type;
    int select_flag;
    int fix_flag;
    double vx,vy,w;
    void *field;
    struct Feature_Pair *f_pair;
    struct Object *next;
    struct Object *group_next;
    struct Object *group;
}    MY_OBJECT;

```

The object is the most important data structure in the system. Each item in the system is an object, including single feature and hierarchy groups. This is realized by the generic pointer "field." This pointer points to the actual data field, such as pin, hole, group, etc. All the objects are stored in an object linked list. For the objects in figure 8, the representation of the system is shown as figure 9. It is a linear linked list. In each object, there is a pointer which points to the next object in the same group, if it is in a group.

There is a "group" pointer in each object which points to the higher level group object. The group object has a pointer in the "field" which points to the head of the lower level object linked list of the group. When we traverse recursively from the head pointer in the "field" of a group object with "group_next" pointer, we can find that we are traversing a tree. We represent figure 8 with group concept as figure 10, it is clearly a hierarchy tree structure. In figure 10, there are two top level objects: group 2 and pin 4. Pin 4 is not in any group. In group 2, the head pointer of the lower level object list points to group1. The "group_next" pointer of group 1 points to the same level object block1. The head pointer of the lower level object list in group 1 points to object hole 2. The "group_next" pointer in object hole 2 points to object hole 3. In this way, we represent the hierarchy data structure of a rigid body.

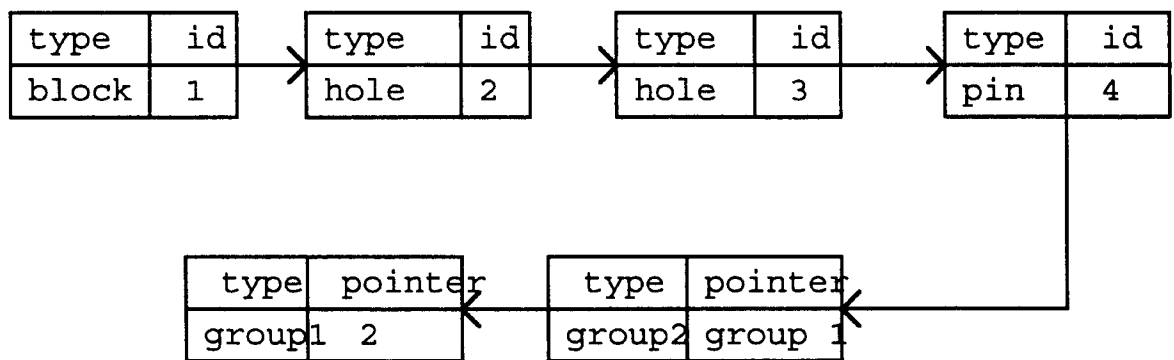


Figure 9. The Object Linked List For Figure 8

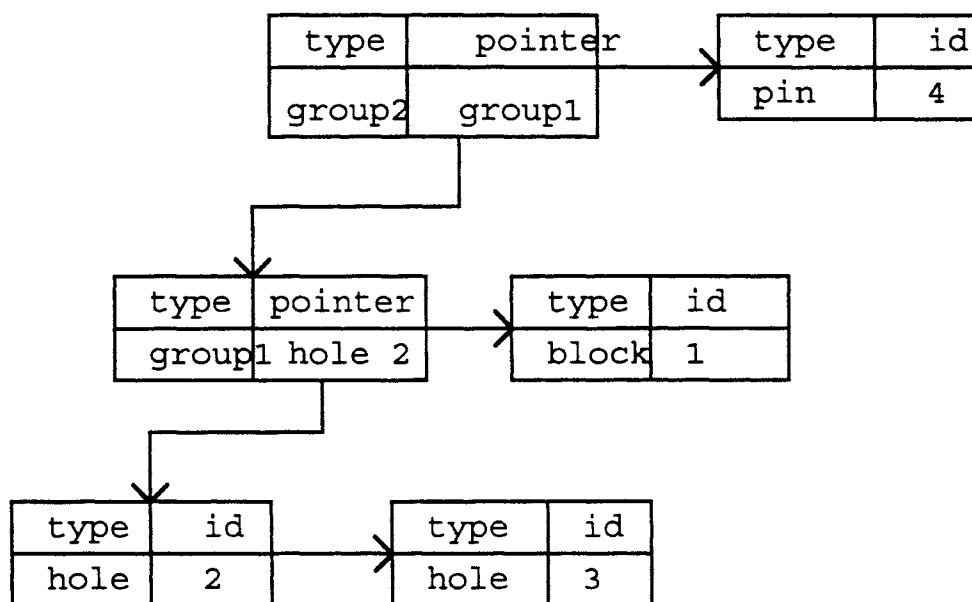


Figure 10. The Tree Data Structure For Figure 8

Concept of Feature Pair

In real world, a mechanical system consists of many components. So the assembly of a system needs to be represented. A data structure represents assemblies in a database that can be divided into two parts. The first part is the data structure used to store topological and geometric information. The second part is the data structure used to store information on how all the components in an assembly are connected. Above we discussed how to represent a component with features. Here, we will propose a feature pair data structure to represent assemblies and implement it in the design system.

In chapter II, we discussed the success and problems of the VIRTUAL-LINK data structure developed by Ko and Lee. Their problems cannot be solved without using other design methodology such as feature based design. But a single feature can only describe the topology and geometry of itself, it is not enough to describe an assembly. So we propose a feature pair methodology to solve these problems.

After analyzing the VIRTUAL-LINK data structure, we can find that a virtual link is actually a pointer which connects the immediate level of the nodes in a tree structure. So it is possible to change the content of the virtual link so that it could include the feature information which we need without changing the hierarchical tree. Next, we will develop a concept of feature pair, which works just as the virtual link, and implement it in our object data structure to develop a hierarchy tree for representing an assembly. The layout of the tree looks almost the same as Lee's hierarchical tree. But we build up a higher level structure on the base of features in the feature based design. This provide a powerful tool to the users and allows them to represent the assembly at a higher level.

Use of the feature captures the designer's intent for component design. The feature pair captures the designer's intent about assembly in the design phase. We define the feature pair as follows:

Feature pair is the complete set of information required to describe the mating features (pin-hole, pin-slot etc.) between two mating components.

In the traditional design procedure, assembly is the first step of a system design. To represent an assembly, we need to provide the following information:

- (1) What kinds of components are in the system.
- (2) The position of each component in the system.
- (3) How components are connected with each other.

Then, according to the assembly, we design the detail of each component.

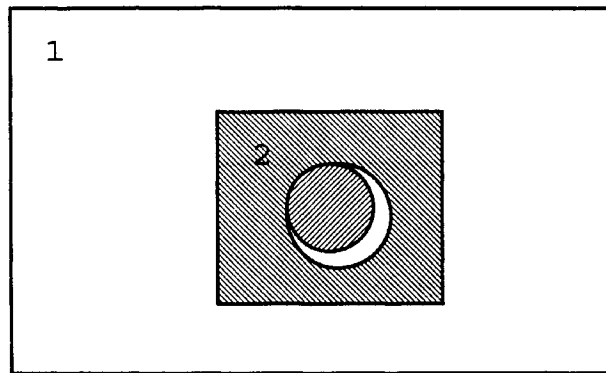
But in today's CAD systems, the procedure is in the reverse order. The designer needs to start with component design. After all components are finished, the assembly could be generated from these components.

Figure 11 (a) shows the assembly of two components, component 1 (b) and component 2 (c). Component 1 is a block with a pin. Component 2 is a block with a hole. In a feature-based design system, the design procedure shows as figure 12.

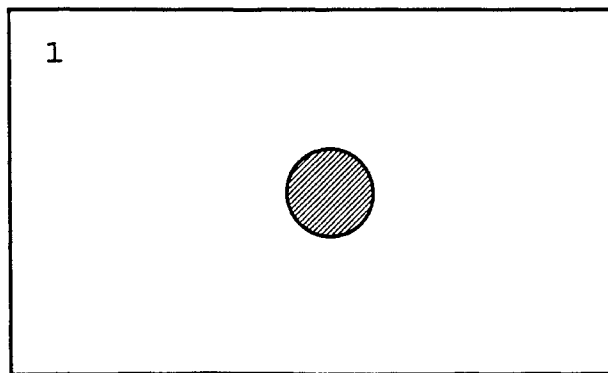
After four features are created, we attach hole 2 to block 1 to form a rigid body, and attach pin 3 to block 4 to form a rigid body. Finally, there are six objects in the system, four features and two rigid bodies (feature groups). The data structure of the system shows as figure 13. Although we know that two groups are connected by the pin and hole, we cannot represent it at this time, because there is no way to represent assembly in present design-with-feature system. We need to move this model to an assembly modeller to represent the assembly.

Now let's discuss how the feature pair design represents the assembly in the design phase. In figure 12, when we create the hole 2, we do know that this hole will mate with the pin 3. So rather than create the single hole, we create hole 2 and pin 3 together. At the same time, we create a 'feature pair' data structure automatically to hold these two features. The procedure shows as figure 14. The most important part is that we represent the assembly automatically although it looks like figure 14 only combine the (b) and (c) in

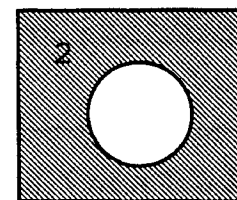
figure 12. When hole 2 is attached to block 1, and pin 3 is attached to block 4, the pointer in the 'feature pair' structure will point to these two components. The data structure shows as figure 15, the difference between figure 15 and figure 13 is very clear here. We can see that the feature pair works just as the virtual link. It links the two components together. The design procedure is almost the same as what we did in the feature based design. But we represent the assembly automatically in a very natural manner when we create the components with the feature pair.



(a) The Assembly of the Two Components

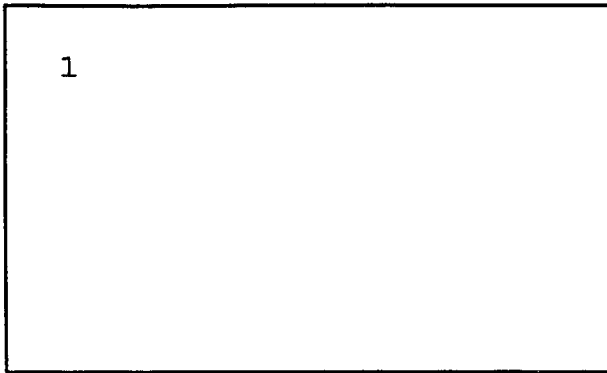


(b) Component 1

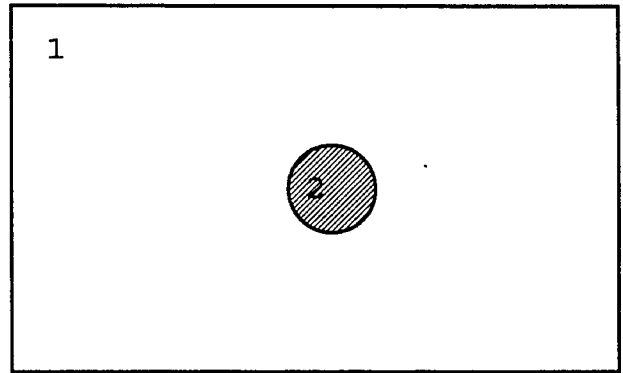


(c) Component 2

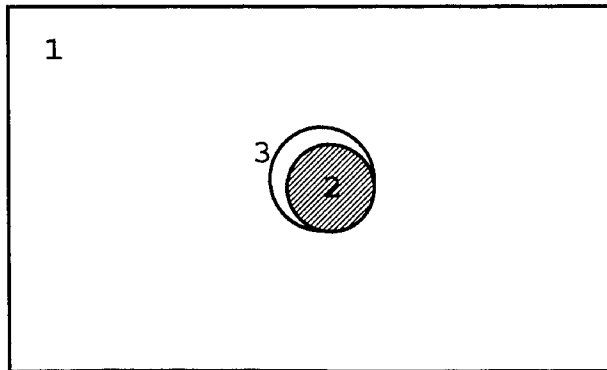
Figure 11 An Assembly with Two Components



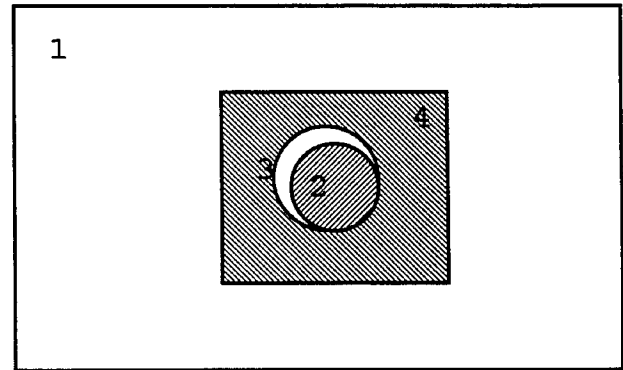
(a) Create a Block 1



(b) Create a Pin on the Block 1



(c) Create a Hole 3



(d) Create Block 4

Figure 12 The Design Procedure for Figure 11

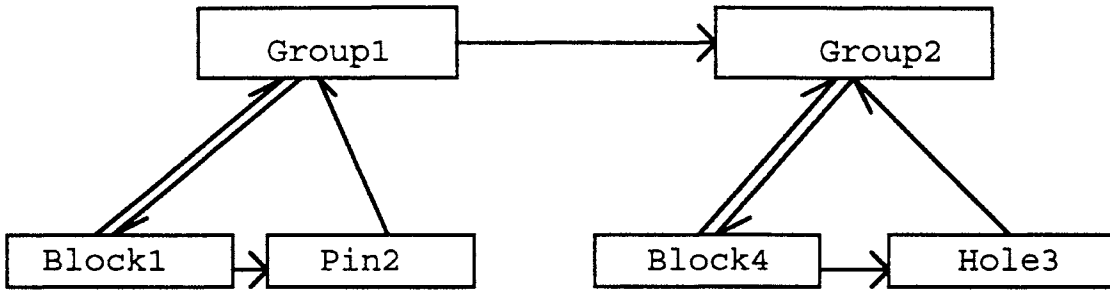
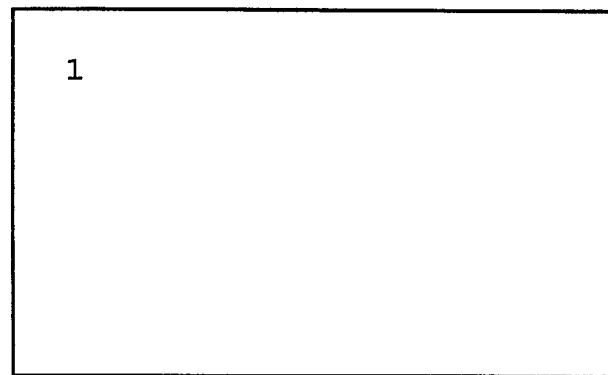
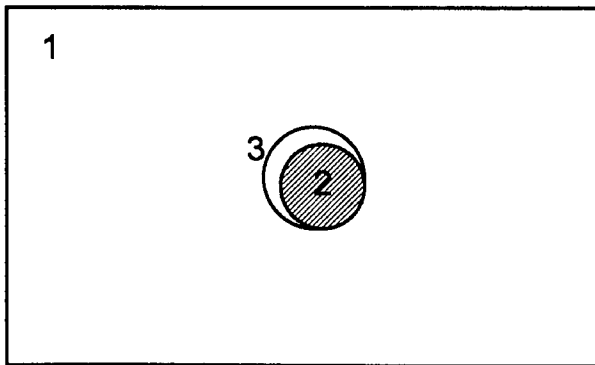


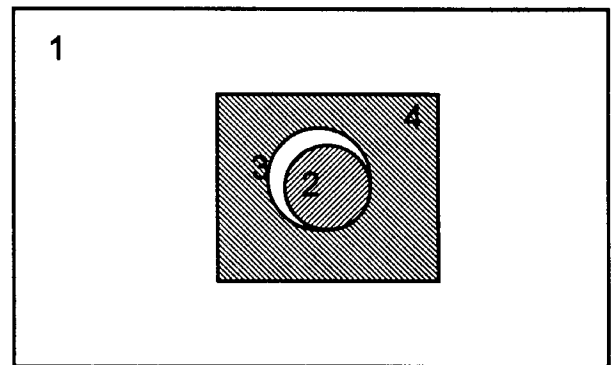
Figure 13 The Data Structure Representation of Figure 11



(a) Create a Block 1



(b) Create a Pin Hole Feature Pair



(c) Create Block 4

Figure 14 The Design Procedure Using Feature Pair Concept

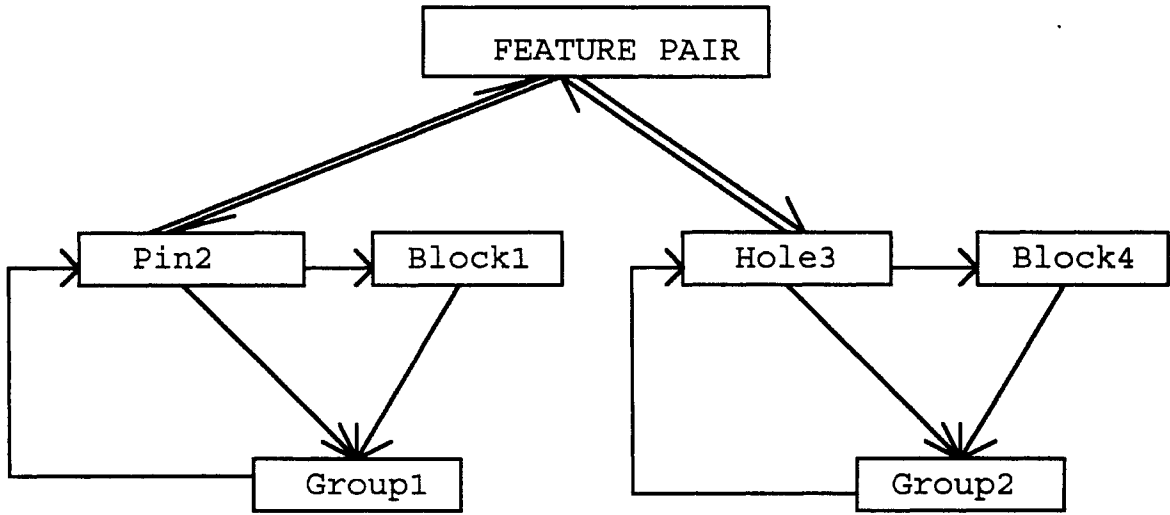


Figure 15 The Data Structure of Figure 11 Using Feature Pair Concept

We define the feature pair data structure as follows:

FEATURE-PAIR

Feature-pair ID.

Feature-pair type (PIN_SLOT, PIN_HOLE, etc.).

The pointer points to the first feature in the pair.

The pointer points to the second feature in the pair.

As we mentioned before, our object data structure is also a hierarchical tree. The feature pair points to the feature in a component, not the component itself. If we traverse the hierarchy from the feature up, the top-level group of it would be the component where the feature attaches. Figure 16 shows a complicated mechanical system with several components. In this figure, "p" stands for feature pair, "c" stands for component (component is the top level group of a feature group hierarchy, it is not in any other group). Due to the space limitation, we cannot demonstrate all the features on the components. Figure 17 is the schematic of the figure 16. It shows the hierarchy of the system. We suppose there are other features that exist in the components. We use "f" to stand for feature, "g" for group. When we consider the assembly of the system, we only want to know how each component is connected. Ignoring all the detail of a component, we get the assembly representation as figure 18.

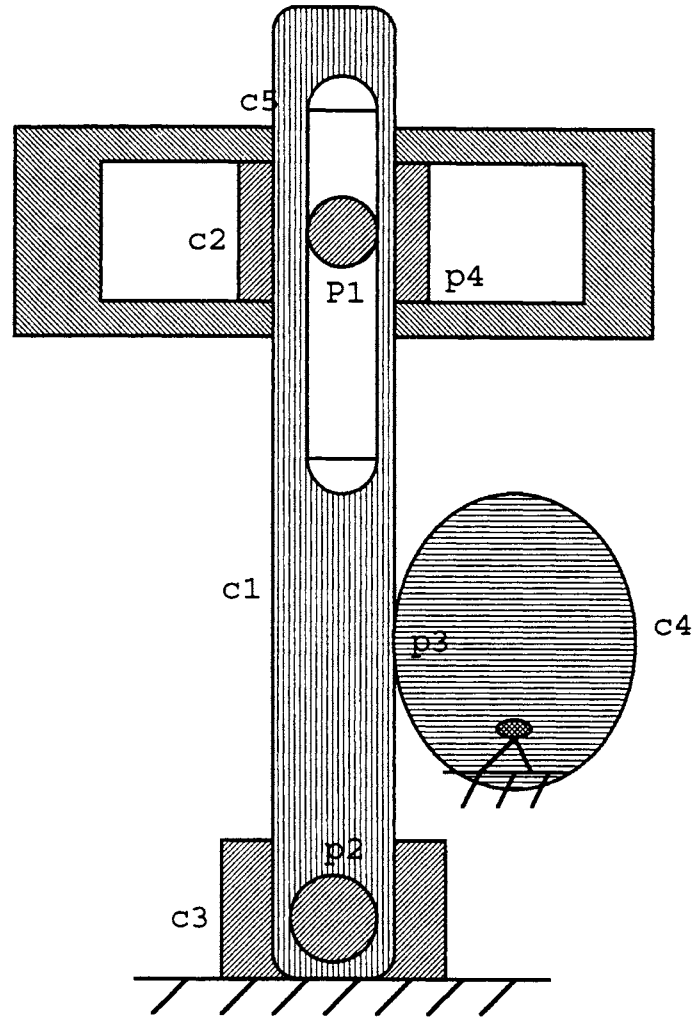


Figure 16 A Complicated Mechanical System with Several Components

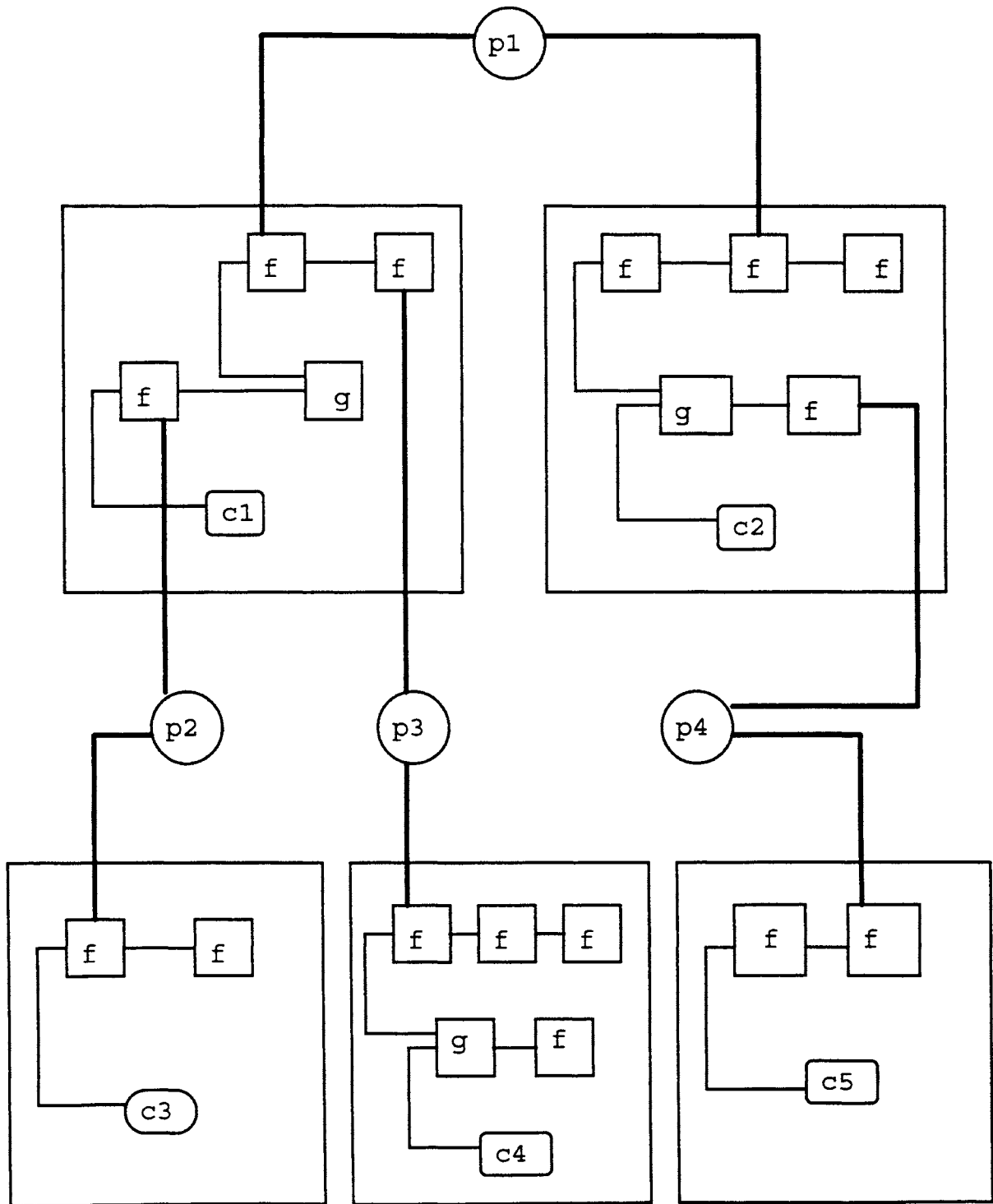


Figure 17 The Schematic for Figure 16

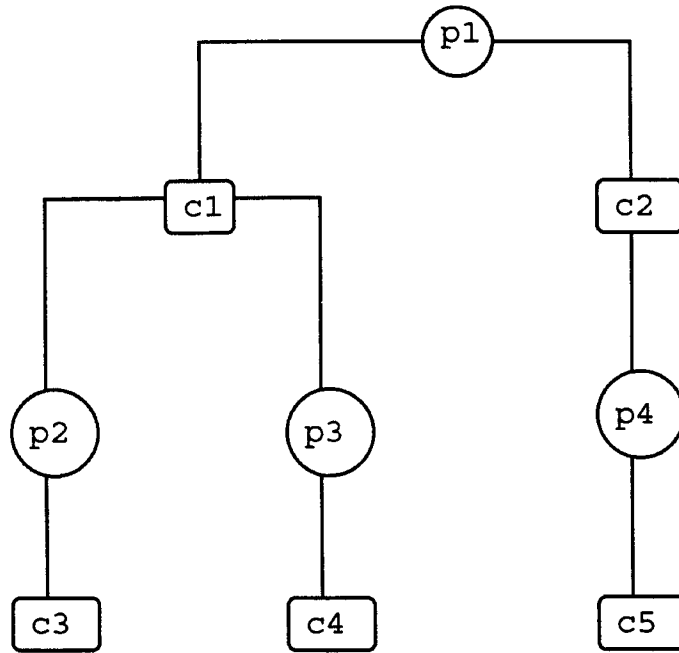


Figure 18 The Assembly Representation of the System

User Interface Design

Event Driven Menu System

The system implements event driven algorithms. In the traditional character-based interfaces, once the application starts, it is always in control. It knows just what kind of input it will allow, and may define exclusive modes to limit that input. For example, the application might ask the user to input with a menu and use the reply to go down a level to a new menu, where the actions that were possible at the previous level are no longer available. Or a text editor may operate in one mode in which keyboard input is interpreted as editor commands and another, in which it is interpreted as data that is to be stored in an editor buffer. In any case, only keyboard input is expected.

In an event driven system, multiple graphic applications may be running simultaneously. In addition to the keyboard, the user can use the pointer to select data, click on buttons or scrollbars, or change the keyboard focus from one application to another. The user can suddenly switch from the keyboard to the mouse, or from one application area to another. Furthermore, as the user moves and resizes windows on the screen, application windows may be obscured or redisplayed. The application must be prepared to respond to any one of many different events at any time.

Event driven programming reduces modes to a minimum, so that the user does not need to navigate a deep menu structure and can perform any action at any time. The user, not the application, is in control. The application simply performs some setup and then goes into a loop from which application functions may be invoked in any order as events arrive.

To realize the event driven, the system provides two application windows. One is the main window which is for display purposes, and the other is a tool box, which contains several groups of icons. When an icon is chosen, a callback function is called, and the system interprets the mouse action according to the current mode. All parameters about

features are figured out by the system through the mouse action. The tool box includes the following groups of icons:

- (1) Create features or feature pairs.
- (2) Select, edit, or delete features or feature pairs.
- (3) Group and ungroup for assembling a rigid body.
- (4) To start the simulation.

The system is designed for a very general purpose. Items are very easy to add on the tool box. To interpret the meanings of these items, the corresponding knowledge base needs to be added. The system is designed toward the direction of the object-oriented methodology. So a feature can be added to the system without any influence to other features in the system.

Select Algorithm

When the user wants to edit an object, he or she needs to tell the system which object needs to be edited. Some systems use the object ID to identify the object. So the user has to remember each object's ID. The user may have a hard time when there are many objects in the system. Once the user knows the ID, he or she needs to input it from the keyboard. This is not a convenient way to make a selection.

In our system, we provide a select-operate algorithm. When we want to operate an object, we first select it. The selection is done by a mouse click. When the mouse is close enough to an object, the object is selected. To do this, the system calculates the distance between the mouse position and the object. If the distance is small enough, the object is selected. Otherwise, all the objects are unselected. The system is smart enough to tell the difference between the distance between a point to an object and the distance between a point and a line. In figure 19, d' is smaller than d , but obviously the object 1 should be selected.

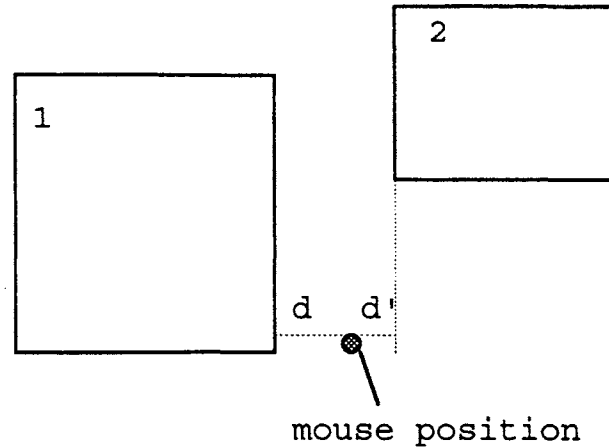


Figure 19. The Selection Algorithm Demonstration

We also provide a multiple selection algorithm. This is used when we want to group some features into a rigid body. When we do multiple selection, we need to select an object (a feature or a group) first, then holding the "shift" key, we select the next object with the mouse. If the "shift" key is not pressed when we select the second object, the first object will be unselected.

The "select_flag" field in the object data structure is used for selection. When an object is selected, the "select_flag" will be set to 1. But if the object is in a group, we always set the flag at the highest level of this group. In figure 19, if object 1 or object 2 is selected, other than set the flag in the object or group 1, we set the flag of group 2. That means the select information will be propagate to the top level.

After we do multiple selection, the selected objects behave as a rigid body. If one object moves, other objects will move together. But it is not a group until the "group" button is pressed. Then a new group object is generated. All the objects selected become the members of the group.

Here we need to address the ungroup operation. When a group is selected and "ungroup" button is pressed, the group object will be deleted from the object list, and the members within this group become "free" objects. But this kind of ungroup only happens for one level. If a member is also a group itself, nothing will change in this group. After the ungroup operation, all these members are multiple selected.

Handles For Editing

After an object (a feature or a group of features) is selected, the handles of it will show up. We can edit the object through these handles. Handles are several sensitive points generated from the object data. Each of them relates with one or more parameters of the object. Different handles are designed for different objects because each object has different data structure. The parameter of the object changes according to the handle's behavior.

When the mouse is close enough to a handle, the handle will be selected. The selected handle will move with the mouse. The system will interpret the coordinate change of the handle to the parameter change of the object. The object will be updated by the new parameter. New handles will be generated from the updated object. The result is when we drag a handle, it looks like we are dragging the object. This gives the user an opportunity to see the editing procedure animated on the screen.

Handles of a Feature. Each feature in the system has its own handle definition.

(1) Handles of a circle or a pin. Because a pin and a hole use the same data structure, the handles of them are the same. As figure 20 shows, there are two handles for the circle. When we drag one of them, the radius of the circle will change with the distance between the mouse position and the center of the circle in x direction.

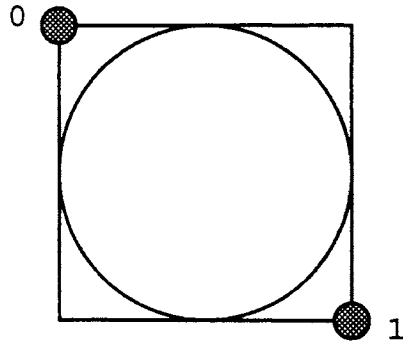


Figure 20. The Handles of A Circle or a Pin

(2) Handles of a block or a slot. A block and slot share the same form of handles. Figure 21 (a) shows the handles for resizing the block. Handle 0, 2, 4, and 6 will resize both length and width of the block. Handle 1 and 5 relate with the length of the block. And 3, and 7 relate with the width of the block. The selected handle is dragged in the mouse direction, and the handle in the opposite side will remain unchanged. Figure 21 (b) shows the handle to rotate a block. This is a submode of the editing. The right mouse can pup up a menu. The rotate entry could be selected for rotating a block. After a handle is selected, the direction of the block will change with the angle between OA and the horizontal line.

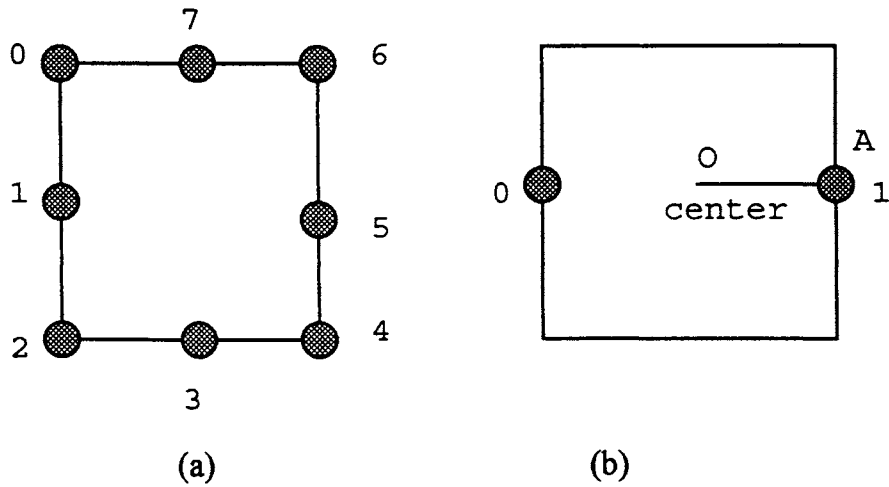


Figure 21. Handles of a Block

(3) Handles of a polygon and sketch. There are two modes to resize a polygon.

Figure 22 (a) shows the handles of the first mode. In this mode, we can scale the polygon. The handles 0, 2, 4, and 6 scale both x and y directions. Handles 1, 3, 5, and 7 scale only one direction. We can get a mirror image if we move handle 5 to the left side of handle 1. Figure 22 (b) shows another mode for resizing a polygon. The handles are the same point as the vertexes of the polygon. This mode allows the user to edit a single vertex of the polygon.

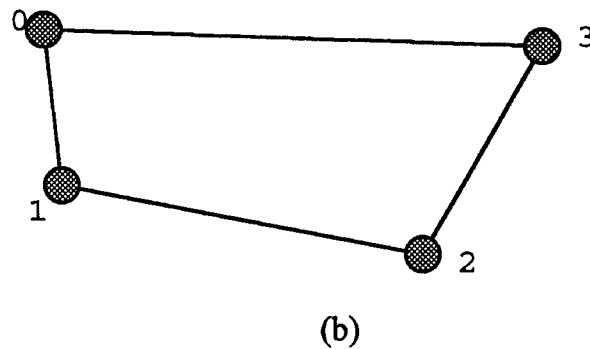
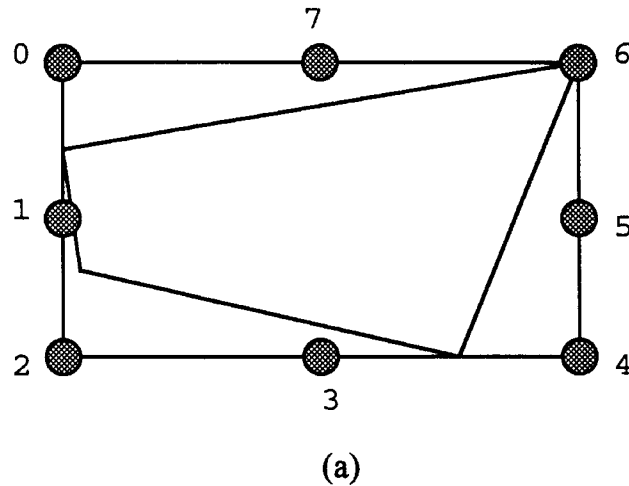


Figure 22. Handles of a Polygon or a Curve

Handles Of A Rigid Body. A rigid body consists of several features. We define a scale operation on the rigid body. That means when a rigid body is scaled, all the features in the body will be scaled. Figure 23 shows the handles of a rigid body. It is a block which

fits in all the features in the body. The handles are stored as the upper left corner (handle 0) and lower right corner (handle 4) in the group data structure.

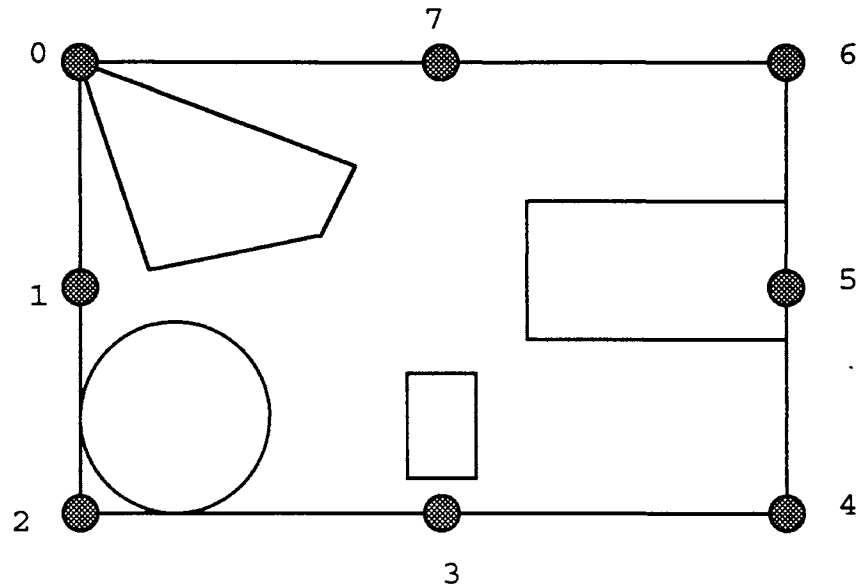


Figure 23. Handles of a Rigid Body

Same as the scaling of a block, handles 0, 2, 4, and 6 scale both x and y directions, Handles 1, 3, 5, and 7 scale only one direction. This works fine for block, slot, and polygon, but there is a problem for the hole and pin. After scaling, the hole and pin should remain in a perfect circle. It is not possible to resize a circle in two directions. So we resize circles according to the x direction.

Modify The Feature Pair

To satisfy the requirement of a feature-pair based modeller, the system must also provide an attractive environment for creating, manipulating and deleting a feature pair. Creating and deleting a feature pair is the same as that of a feature. However, the modifying of a feature pair is a little tricky.

In modifying a feature, the user's intent is to change the parameter of a feature and keep its geometry and topology remaining unchanged. And the user's intent in modifying a

feature pair is to change the parameters of the feature pair or the position of it and keep the remaining mating situation unchanged.

Because we try to use the mouse as the only input device, we need to give a correct interpretation for each mouse action. In modifying a feature pair, there are several possible interpretations for one mouse action. So we need to discuss each possibility in detail. But in any situation, we need to remember that the mating situation of the feature pair must be maintained.

Change The Parameter of a Feature Pair.

The handle discussed above is used here. And we also use the select-edit method for editing feature pairs. The difference is that when a feature in a feature pair is selected, the other feature in the feature pair will also be selected. When we edit a feature, to maintain the mating situation of the feature pair, the corresponding parameter of the other feature needs to be changed accordingly. We take the figure 12 as an example. After strain analysis, it might be necessary to increase the radius of the pin. If we do not change the radius of the hole, the mating restraint will be violated. Also, the pin could not be too large to exceed the block 4 because it is not a valid mating pair if the pin is larger than the hole. The interference between features is not allowed in the system. In a pin-slot feature pair, we also need to take care of the interference problem. The diameter of the pin should be the same as the width of the slot. And the length of the slot should not be smaller than the width of it.

Change The Position Of A Feature Pair.

Change The Position Of A Feature Pair Whose Feature Is Not In A Group.

There are two possibilities that exist here. The first is that the relative movement is allowed between the two features in the feature pair. In figure 24, slide 1 and square slot 2

are a slide-slot feature pair. At the beginning, the user hopes the bar AB could start at the angle a . Then he or she might want to decrease a . We need to move slide 1 to the right in slot 2. At this point, the system should know that slide 1 can only move in the slot direction. And right side of slide cannot exceed GF, left side cannot exceed HE. So when relative movement is allowed in a feature pair, one feature could be moved with some restraint. After it is moved, the interference should be checked. To do this modification, we add another handle at point D in the slider. When this handle is moved, the slider is moved with the described restrain.

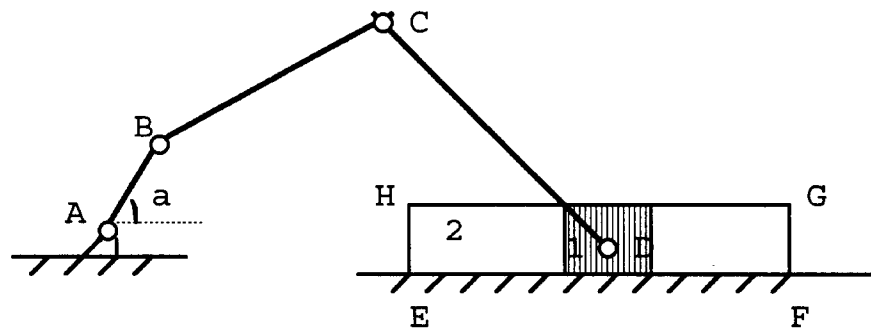


Figure 24. Relative Movement in a Feature Pair

The second possibility is that relative movement is not allowed between the two features. Figure 25 shows a pin-hole feature pair. The hole 1 should be on block 3. We need to move the hole to the dotted circle position on block 3, then attach hole 1 to block 3. For mating condition, the pin 2 needs to be moved together with the hole 1. So when relative movement is not allowed in a feature pair, the two features in the feature pair are always moved together.

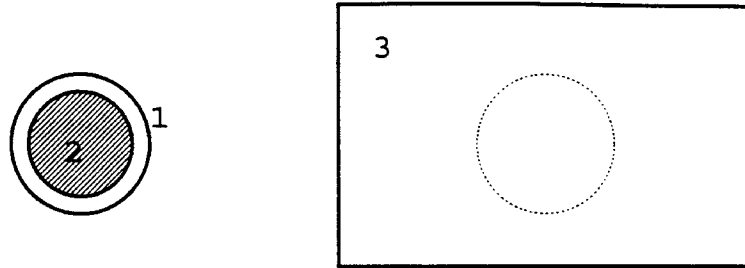


Figure 25. Move a Feature Pair with no Relative Movement

Change The Position Of A Feature Pair Whose Feature Is In A Group. When features in a feature pair are in groups, we need consider not only the movement of the feature pair but also the other features in the same group. There are two possibilities here. When two features are all in groups, that means the features have been already in rigid bodies. After moving them, we need to keep the features in the right position in the rigid body, and the two rigid body are in the right mating position. So when we move them, we move all the rigid bodies together. If there are other feature pairs in the rigid body, they will be moved together. In figure 26, there are two holes in block 1: hole 2 and hole 3. Both block 4 and block 5 have a pin attached to them. Hole 3 and pin 7 are a feature pair. Hole 2 and pin 6 are a feature pair. When we move the feature pair 3 and 7, all of the objects in figure 26 should be moved together. This is the only way to satisfy the mating condition in the system.

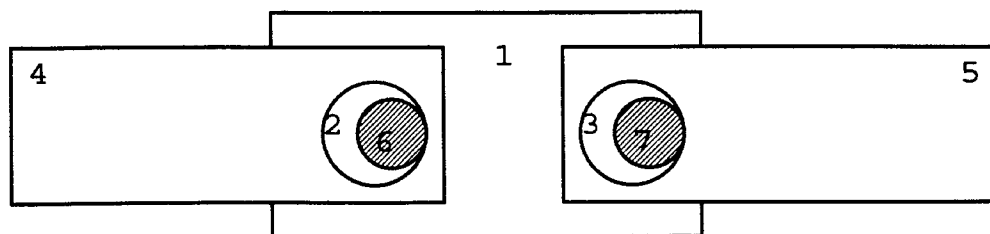


Figure 26. Move All Connected Feature as a Rigid Body

Another possibility shows as figure 27. Pin 2 and hole 3 are a pin-hole feature pair. Hole 3 is attached to block 4. The user wants to move hole 3 to the dotted position in block 1. We need to move pair 2, 3 together and keep block 1 unmoved.

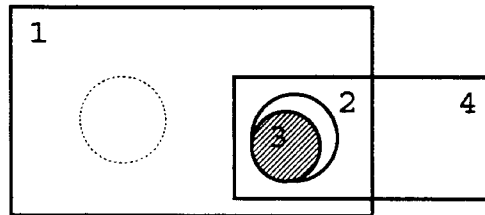


Figure 27. Move a Feature Pair in a Rigid Body

Visual Effect of the Graphics

In a general mechanical system, there are at least two components. Each component consists of several features. In the real world, we can distinguish them by shape, color, and material under the real light. On the screen, how to give the user a better visual effect is a problem.

We have several ways to solve this problem: shading, different colors, and line styles. Another alternative is to flash the selected group or feature pair. It works only if one group or feature pair is selected. Although shading is the best way, it is not possible at this time because the light model is not implemented in the system. We use different colors to represent different groups, and we use dash lines to represent feature pairs. Black is the default color in the system. The user can change the color of a selected object or objects in a pup up window. The window also provides a slider which allows the user to change the stiffness of the same object(s).

CHAPTER IV

IMPLEMENT FEATURE PAIR AND DISTRIBUTED COMPUTATION IN DYNAMIC SIMULATION

A simulation of a mechanical system can act as a cost effective test bed for final designs. Due to changes in the kinematics constraints, many mechanical systems are described by discontinuous equations of motion. In our system, we use a simple dynamic model which can detect constraint changes automatically. This model was discussed in chapter II. But Hong's system is designed to handle a specific mechanical problem. In our system we use the model to handle general problems. The calculation is very complicated. So we try to find ways to minimize the number of calculations we need to do in the simulation.

Use Feature Pair To Speed Up The Simulation

In chapter II, we discussed the dynamic model developed by Hong. Here we try to use the feature-pair concept to minimize the calculation needed for the dynamic calculation. Figure 28 shows a simple mechanical system including 6 objects. The spring initials a movement on the 1. This starts simulation of the system. For each time step, the solver will check object 1 with object 2, 3, 4, 5, and 6. Check object 2 with 3, 4, 5, and 6. Check 3 with 4, 5, and 6. Check 4 with 5, and 6. Finally, check 5 and 6. If the system has many objects, many calculations need to be done. This slows down the simulation speed.

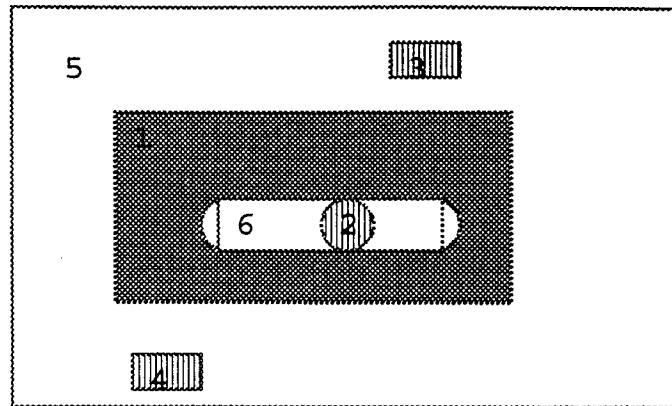


Figure 28. A Simple Mechanical System

We can see that pin 2 and block 5 are a rigid body, and so is slot 6 and block 1. They cannot intersect with each other. Before simulation, the system will check illegal intersection automatically. That means we do not need to calculate the intersection between objects in a same rigid body.

We propose a feature pair concept in chapter III. It could also be used here to speed up the simulation. In figure 29, hole 2 is in block 1. It is a feature pair with pin 3. We need to check the intersection between the hole 2 and pin 3. It is the only possible intersection for pin 3 and hole 2. So we can conclude that if a feature is in a feature pair, the only possible intersection for it is the intersection within the feature pair. The following tables shows how the calculations are reduced.

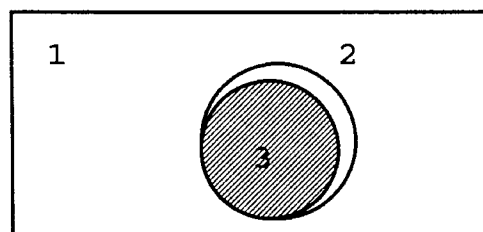


Figure 29. Speed Up the Simulation with Feature Pair

In the following tables, "O" stands for a calculation reduced from the previous table. From table I to table II we decrease the calculation from 15 to 13 using the concept

of rigid body. In table III, the calculation decreases to 7 using the concept of a feature-pair. So we can expect to speed up the simulation twice when we use the feature-pair data structure developed.

TABLE I
NUMBER OF CALCULATION IN GENERAL METHOD

object	1	2	3	4	5	6
1		X	X	X	X	X
2			X	X	X	X
3				X	X	X
4					X	X
5						X
6						

TABLE II
NUMBER OF CALCULATION WITH RIGID BODY CONCEPT

object	1	2	3	4	5	6
1		X	X	X	X	
2			X	X		X
3				X	X	X
4					X	X
5						X
6						

TABLE II
NUMBER OF CALCULATION WITH FEATURE PAIR CONCEPT

object	1	2	3	4	5	6
1			X	X	X	
2						X
3				X	X	
4					X	
5						
6						

Distributed Computation For Simulation

As we know, Silicon Graphics workstation is very powerful for graphics. But the mathematics calculation speed is not fast. Having access to RS6000 which has high float point speed, we want to use Silicon Graphics workstation for displaying, use RS6000 for calculation, and use network to connect them. The network communication overhead will override the speed up and even slow down the simulation than running them locally although RS6000 is much faster than Silicon Graphics workstation.

From table I, table II, and table III, we can see that the position of an object is calculated separately from the position of all objects' previous position. So it is possible to run several copies of dynamic solver on several RS6000. Each solver takes care of one or more objects. After the calculation is finished, it sends back the next position of calculated objects to Silicon Graphics workstation. After Silicon Graphics workstation gets all the results from the solvers, it will display new positions of objects on the screen, and send them to all solvers.

There is more overhead here. Suppose we have n objects in the system. If we do the calculation locally, the total number of calculation is

$$\frac{n^2 - n}{2}$$

If we use n solvers, a solver for an object, the total number of calculation are

$$n*(n-1)$$

that is twice of the local operation. This is not a problem because n calculations is performed at the same time. So the actual number of calculation is

$$n-1$$

which is smaller than the local calculation. The most important problem comes from the data communication speed. We need to send the new position of each object n times. The

transferring time is a main factor here. Now we will discuss how much speed up we can get with this distributed calculation.

Suppose we have n objects in a system. Average calculation speed for collision of two objects on Silicon Graphics workstation is T . RS6000 is m times faster than Silicon Graphics workstation, and each RS6000 has the same speed. Because the concept of rigid body and feature pair affect both local and distributed method, we ignore their influence. Then the total time needed for local method should be

$$\frac{n^2 - n}{2} * T$$

We assume that the transfer speed for a double precision number is t . For 3 number, coordinate of centroid and rotation angle, is $3t$. We send the new position back, and $n-1$ objects' new position to the solver. So total transfer time is

$$3n^2t$$

The number of calculation on each RS6000 is $n-1$. The calculation time is

$$(n-1)*T/m$$

We get the total time for distributed method is

$$3n^2t + (n-1)*T/m$$

We want distributed method is faster than local method, we need

$$3n^2t + (n-1)*T/m < \frac{n^2 - n}{2} * T$$

Suppose $m \gg n$, we ignore $(n-1)*T/m$, then get

$$3n^2t < \frac{n^2 - n}{2} * T$$

So

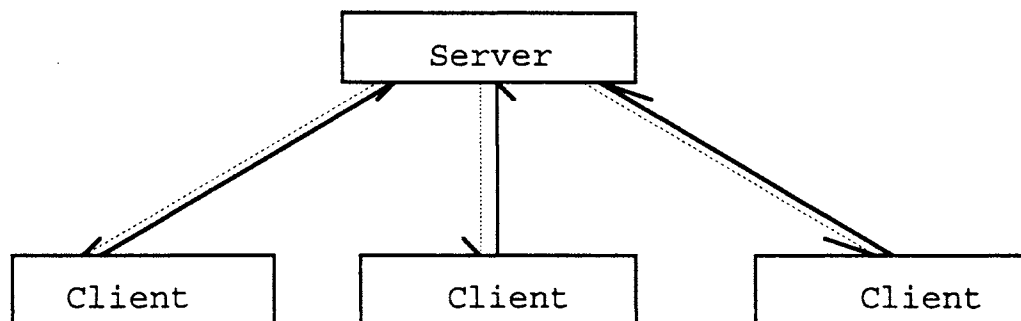
$$\frac{t}{T} < \frac{n-1}{6n}$$

We have

$$t < \left(\frac{1}{6} - \frac{1}{6n}\right)T$$

So we know if the data transfer speed is fast enough, we can speed up the simulation with distributed method.

In the system, we use a server-client model with BSD socket as the communication protocol. Silicon Graphics workstation works as server, and RS6000 works as client. The server connects with several clients by socket. Figure 30 shows the client-server model.



..... current position of all objects
 ——— the new position of the calculated object

Figure 30. Server-Client Model

When the connection between server and clients is established, the server will transfer the current position of objects to each client and specify which object (or objects) the client should calculate. After all clients send back the results, the server displays the new position and sends these new positions to each client. This is shown as figure 31.

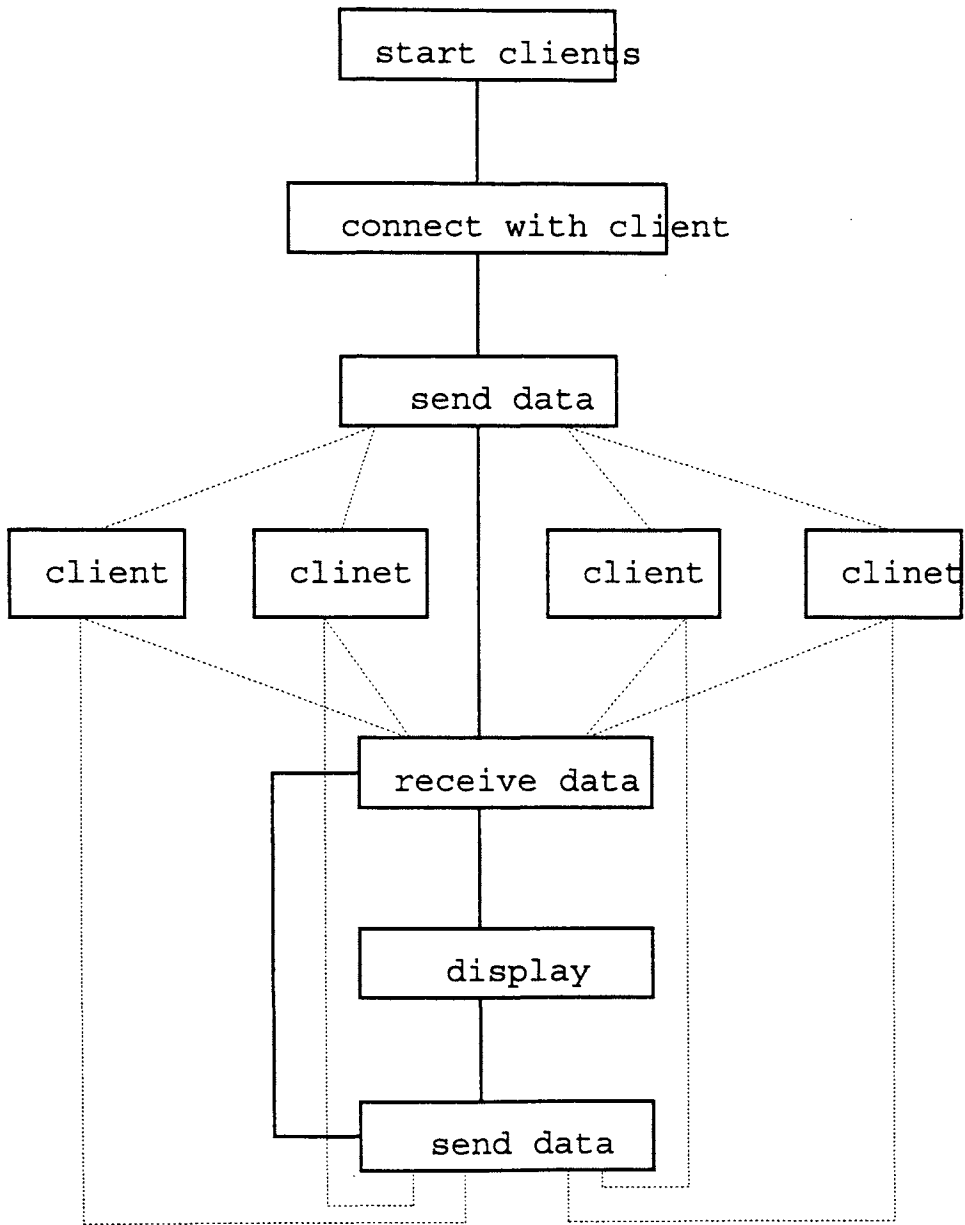


Figure 31. Flow Chart of the Server

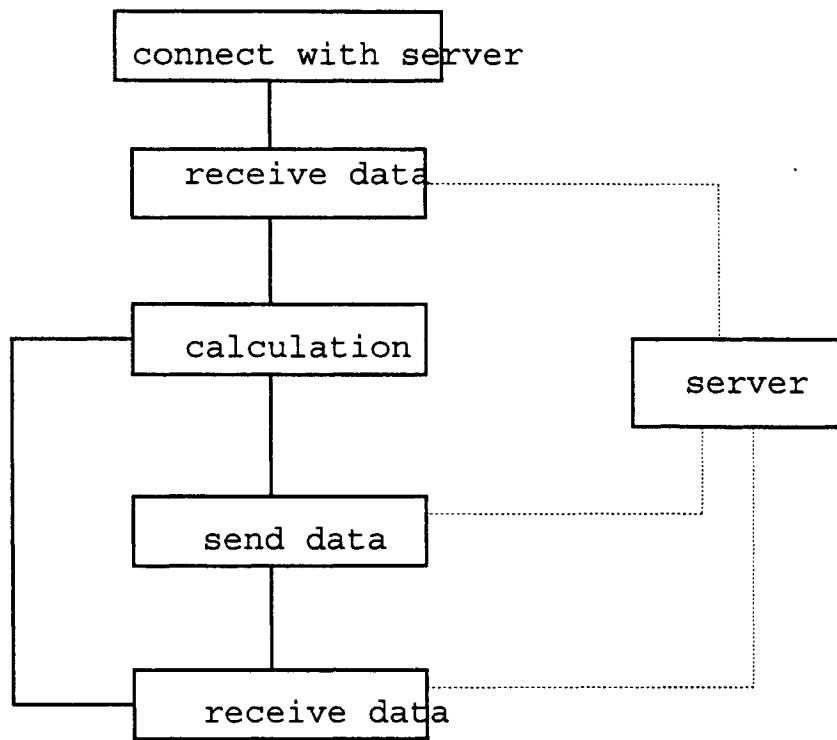


Figure 32. Flow Chart of the Client

For the client, after the connection is established, it gets the current position of all objects. Knowing which object it should calculate, it calculates the new position of the object. Then it sends it back to the server. This is shown as figure 32.

At the beginning of the simulation, the server starts the clients on RS6000 remotely. For security reasons, there is no password in the program explicitly. The system will go to .netrc file for checking the machine name, login name, and password. To avoid failure to start the client, several protection procedures are added here.

We use the "rexec" function to start the client remotely. But "rexec" returns at once without blocking. If one client fails to start due to a machine crash, the server will terminate. However, the started clients still try to connect with the server which is not exist. This gives the network a heavy load and could crash the whole network. To solve this problem, we use a timeout mechanism. In stead of trying to connect with the server all the time, the client tries to connect with the server for several times every three seconds. If the connection fails for more than four times the client will terminate automatically. This avoids deadlock and reduces the load on network.

After the communication is finished, the kernel will not release the binding address immediately. If we use a specific port number and start a simulation immediately after a simulation is finished, the system will indicate the address that has been used, and the process will be terminated. So we need to try a new port number. If it works, the port number will be pass to the client as a parameter.

CHAPTER V

CONCLUSION AND RECOMMENDATION

Conclusion

This system is developed for the planar mechanical system design. It is an integrated design environment which is very convenient to use. The user can come back and forth among the rigid body design, assembly design, dynamic simulation, and distributed computation interactively. This gives the user much flexibility in the iteration design process.

In the system, we use features to capture the designer's intent for topology and geometry on any single component. The feature-pair concept we developed successfully captures the designer's intent about assembly. It generates assembly hierarchy automatically at the design phase and allows the user to modify any feature even after the assembly has been constructed. In the dynamic simulation, the feature pair works well to reduce the number of intersection calculations and speeds up the simulation.

To support the feature pair based design, we developed an attractive graphical user interface. The select-edit method and editing handle in this interface makes it possible to use a mouse as the only input device. This minimizes the user's input and reduces the possibilities of the user's error. The event driven menu system integrates all operations into one tool box. The icons used in the tool box provide a better visualization for the user.

The dynamic model we use is simple to implement and works well in most situations. It successfully detects constraint changes automatically.

The system also has some drawbacks. Because we hard code the constraints of a feature, the user cannot define new constraints for the feature pairs. Although we allow the user to define new feature pairs using the features, the constraints of these new feature pairs cannot be defined by the user. The feature pair works well in a complete system design. In an incomplete system, a feature in a feature pair may not be attached to any rigid body. This may cause problems in the simulation.

We use distributed computation technology to speed up the dynamic simulation. It is implemented on RS6000 through the BSD Socket. It works, but the result is not satisfactory. The main problem is that the overhead of the communication speed slows down the simulation.

Recommendation

In our system, only a few features and feature pairs are provided to the user and those are not enough. We need to develop a feature and feature pair database, which can be customized and accessed by the user.

Our system is designed for a 2D problem. We need to combine it with a solid modeller so it can handle a 3D problem.

The dynamic model used in the system is not the best one. Our system would be more powerful if we connect it with a solver which implements the Gilmor and Cipra's algorithm.

For distributed computation, we need a scheduler to assign jobs to each RS6000, so their load could be balanced and work effectively. The major bottle neck to slow down the simulation is the communication speed. Maybe we can run our system on a true parallel architecture, such as Squent or hypercube computer.

REFERENCES

- Cunningham J. J. and Dixon, J. R. (1988). Designing With Features: The Origin of Features, Computers in Engineering, Transactions of the ASME, 1, pp. 237-243.
- DeFazio, T. L., Edsall, A. C., Gustavson, J. A. and Hernandez, J. A. (1990) A Prototype of Feature-Based Design for Assembly. Advances in Design Automation. 23(1)pp. 9-16.
- Gilmore, B. J. and Cipra, R. J. (1991). Simulation of Planar Dynamic Mechanical Systems With Changing Topologies-Part 1: Characterization and Prediction of the Kinematics Constraint Changes, Transactions of the ASME, 113, pp. 70-76.
- Gilmore, B. J. and Cipra, R. J. (1991). Simulation of Planar Dynamic Mechanical Systems With Changing Topologies-Part 2: Implementation Strategy and Simulation Results for Example Dynamic Systems. Transactions of the ASME, 113, pp. 77-83.
- Henderson, M. R. and Chang, G. J. (1988). FRAPP: Automated Feature Recognition and Process Planning From Solid Model Data. Computers in Engineering 1988-proceedings, pp. 529-536.
- Hirschtick, J. K., Gossard, D. C. (1986). Geometric Reasoning for Design Advisory Systems. Proceedings of the 1986 Computers in Engineering Conference. pp. 263-270.
- Hong, X. (1991). Concepts for Computer-Aided Preliminary Design and Modeling of Assemblies with Moving Parts. Oklahoma State University Master Thesis.
- Hummel, K., Brooks, S. (1986). Symbolic Representation of Manufacturing Features for an Automated Process Planning System. Proceedings of Winter Annual Meeting of the ASME. pp. 233-243.
- Irani, R. K., Saxena, M. and Finnigan, P. M. (1990). Boundary-Based Feature Modeling Utility. Computers in Engineering 1990 Proceeding. pp. 45-51.
- Ko, Heedong and Lee, Kunwoo (1987). Automatic Assembling Procedure Generation from Mating Conditions. Computer Aided Design, 19(1), pp. 3-9.
- Kumar, B., Anand, D. K. and Kirk, J. A. (1988) Knowledge Representation Scheme for an Intelligent Feature Extractor. Computers in Engineering 1988-proceedings, pp. 543-550.

- Lee, Kunwoo and Gossard, G. C. (1985). A Hierarchical Data Structure for Representing Assemblies : Part 1. Computer Aided Design, Jan. 1985, pp. 15-19.
- Mullender, Sape (1989). Distributed System. ACM Press, New York.
- Pratt, M. J. (1988). Synthesis of an Optimal Approach to Form Feature Modeling. Computers in Engineering 1988-proceedings, pp. 263-273.
- Shah, J. J., Rogers, M. T. (1988a). Feature Based Modeling Shell: Design and Implementation. Computers in Engineering 1988-proceedings, pp. 255-261.
- Shah, J. J. and Rogers, M. T. (1988b). Expert Form Feature Modeling Shell. Computer Aided Design, 20(9), pp. 515-524.
- Shah, J. J. and Bhatnagar, A. S. (1989). Group Technology Classification from Feature - Based Geometric Models. Manufacturing Review, 2(3), pp. 204-213.
- Shah, J. J., Tadepalli, Ravi (1992). Feature Based Assembly Modeling. Computers in Engineering, 1992 : Proceedings of the 1992 ASME International Computers in Engineering Conference and Exposition. pp. 253-260.
- Shah, J. J. (1991). Assessment of Features Technology. Computer Aided Design, 23(5), pp. 331-341.
- Smith, D. A. (1973). Reaction Force Analysis in Generalized Machine Systems. Journal of Engineering for Industry, May 1973, pp. 617-623.
- Turner, G. P. and Anderson, D. C. (1988). An Object-Oriented Approach to Interactive Feature-Based Design for Quick Turnaround Manufacturing. Computers in Engineering 1988-proceedings, pp. 551-555.
- Unger, M. B. and Ray, S. R. (1988). Feature-Based Process Planning in the AMRF. Computers in Engineering 1988-proceedings, pp. 563-569.

VITA 2

PENGFEEI CAI

Candidate for the Degree of

Master of Science

Thesis: A USER INTERFACE FOR FEATURE-PAIR BASED DESIGN AND
ANALYSIS OF MECHANICAL ASSEMBLY.

Major Field: Mechanical Engineering

Biographical:

Personal Data: Born in Beijing, P.R.China, May 21, 1967, the son of Qigui Cai and Yuehua Jiao.

Education: Graduated from No. 4 High School, Beijing, P.R.China, in July 1985; received Bachelor of Engineering Degree from Tsinghua University, Beijing, P.R.China, in August, 1990; completed requirements for the Master of Science degree at Oklahoma State University in May, 1994.

Professional Experience: Assistance engineer in China Daheng Co.; Research Assistant, School of Mechanical and Aerospace Engineering, Oklahoma State University, January, 1992, to December, 1993.